BODOR TIBOR

A COMMODORE 64 PROGRAMOZÁSÁNAK GYAKORLATA

SOROS LEMEZÁLLOMÁNYOK

2

SZÁMALK

A COMMODORE–64 PROGRAMOZÁSÁNAK GYAKORLATA II.

Soros lemezállományok

COMMODORE PROGRAMMING IN PRACTICE II.

Sequential Disk Files

TIBOR BODOR

COMMODORE PROGRAMMING IN PRACTICE

SEQUENTIAL DISK FILES

Second volume

COMPUTING APPLICATIONS AND SERVICE CO. BUDAPEST, 1986

BODOR TIBOR

A COMMODORE 64 PROGRAMOZÁSÁNAK GYAKORLATA

SOROS LEMEZÁLLOMÁNYOK

Második kötet

SZÁMÍTÁSTECHNIKA-ALKALMAZÁSI VÁLLALAT BUDAPEST, 1986

COMMODORE–64 SOROZAT

Lektorálta: *Danis Miklós*
Supervised by *Miklós Danis*

ISBN 963 553 115 x (összkiadás)
ISBN 963 553 121 4 (II. kötet)

Tartalom

Előszó 9

I. RÉSZ – LEMEZKEZELÉS 11

Lemezkezelési alapismeretek 13

A lemezkezelés sajátosságai – Állománykezelő utasítások – Adatkezelés soros állományokban – Rekordok kezelése a soros állományon belül – A lemezkezelési szabályok összefoglalása

A lemez fizikai sajátosságai 43

A 18-as sáv 0-ás szektorának tartalma – A 18-as sáv 1-es szektorának tartalma – A 18-as sáv összes többi szektorának tartalma – A tartalomjegyzék egy bejegyzésének tartalma – A lemez nevének ellenőrzése programból – A programnevek kigyűjtése a tartalomjegyzékből – Saját tartalomjegyzék készítése

Egyéb lemezkezelési tudnivalók 67

Kétlemezes funkciók – Több lemezegység kezelése – A lemezműveletek sikerességének lekérdezése – Megjegyzések a lemezkezeléshez – A soros állományokra vonatkozó hibaüzenetek – A COMMODORE adatállományai

II. RÉSZ – SOROS ÁLLOMÁNYOK 75

Alapműveletek soros állományokkal 77

Soros állomány – Létrehozás – Feldolgozás – Bővítés – Másolás – Összemásolás – Törlés – Átnevezés – Szűkítés – Módosítás – Felülírás – Lekérdezés – Válogatás

Rendezett állományok 123

Keresés – Párosítás – Karbantartás – Rendezett felvitel – Csoportonkénti feldolgozás – Összefésülés

Adatfeldolgozó rendszerek 153

Keretrendszer – Rendezetlen karbantartás – Feldolgozási menetek – Gyorsított karbantartás

Utószó 177

Irodalom 179

Tárgymutató 181

Contents

FOREWORD 9

PART I 11

DISK HANDLING 13
disk handling, file handling, data handling, record handling, summary

PHYSICAL ORGANIZATION OF THE DISK 43
disk header, directory, block availability map, directory entries, getting information from the directory

MORE ABOUT DISK HANDLING 67
using dual drives, using two or more disk units, accessing the command channel, summary, error messages, the COMMODORE disk files

PART II 75

SEQUENTIAL FILES 77
creating, processing, appending, copying, concatenating, scratching, renaming, deleting, modifying, rewriting, searching, selecting

ORDERED FILES 123
searching, matching, updating, creating, control breaking, collating

DATA PROCESSING SYSTEMS 153
driver module, updating, processing phases, optimizing

EPILOGUE 177

REFERENCES 179

INDEX 181

Előszó

Mindenekelőtt köszönjük kedves Olvasóinknak az első kötettel kapcsolatos észrevételeiket, éspedig nemcsak az elismerő véleményeket, hanem a kritikákat is. Természetesen a Kiadóval karöltve mindent megteszünk, hogy sorozatunk további kötetei még inkább megnyerjék az olvasóközönség tetszését.
Noha adatállományok kezelésével elsősorban azok foglalkoznak, akik már túl vannak a kezdeti „szárnypróbálgatásokon", könyvünk megírásakor továbbra is ahhoz a vezérelvhez tartottuk magunkat, hogy az a programozással csak éppen ismerkedő olvasók számára is érthető legyen, továbbá, hogy kizárólag olyan programokat tartalmazzon, amelyeket az olvasó maga is lefuttathat és kipróbálhat.
Az állományok kezelése, és egyáltalán a lemezkezelés, nagyobb arányban igényel gyakorlati tapasztalatot, mint lexikális ismereteket. Ezért azt javasoljuk, hogy a bemutatott lehetőségeket az olvasó a könyv tanulmányozásával párhuzamosan azonnal próbálja is ki saját gépén; majd a könyv végére érve az így megszerzett ismereteinek elmélyítése céljából a lemezre kimentett mintaprogramokat különféleképpen módosítgatva minél több változatban próbálja ki, és elemezze a módosítások hatását. Ezáltal eljuthat arra a szintre, hogy önállóan is tudjon soros adatállományokat lemezen létrehozó és feldolgozó programokat írni.
Minthogy a COMMODORE gépet nem adatfeldolgozásra tervezték, könyvünk elsődleges célja, hogy az olvasó a COMMODORE-t mint tanulógépet használva elsajátíthassa a soros adatfeldolgozás alapelemeit.

A szerző

Budapest, 1986. május

I. RÉSZ
LEMEZKEZELÉS

Lemezkezelési alapismeretek

A lemezkezelés alapműveleteit sorozatunk első kötetében ismertettük, de ott csak a programokkal kapcsolatos lemezkezléssel, programbetöltéssel, kimentéssel stb. foglalkoztunk. Ezeket az ismereteket most ki kell egészítenünk a soros állományokra vonatkozó lemezkezelési információkkal. Eközben nem kerülhetjük el a leglényegesebb elemi tudnivalók felidézését sem.
A lemezműveletek végrehajtására lemezkezelő parancsokat kell kiadnunk. Ezeket három kategóriába sorolhatjuk:

— az egész lemez kezelésére vonatkozó parancsok
  NEW : lemez formázása, újraformázása
  VALIDATE : lemez tömörítése
  INITIALIZE : lemezegység helyreállítása
— csak programokat kezelő parancsok
  LOAD : program betöltése
  SAVE : program kimentése
  VERIFY : program ellenőrzése
— programokat és állományokat egyaránt kezelő parancsok
  COPY : program, állomány másolása
  RENAME : program, állomány átnevezése
  SCRATCH : program, állomány törlése

A programkezelő LOAD, SAVE, VERIFY parancsok közvetlenül kiadhatók, a többit a parancs-csatornára kell kiküldeni a PRINT utasítással. A csatornát előzetesen az OPEN utasítással meg kell nyitnunk. (Megjegyezzük, hogy a parancsok a PRINT helyett ebben az OPEN utasításban is szerepelhetnek.)
Emlékeztetőül vegyünk elő egy üres lemezt, és hajtsuk végre az alábbi műveletsorozatot. A lemezre később még szükségünk lesz, mert ezt használhatjuk majd a soros állományok kezelésének kipróbálására.
A lemez új, formázzuk. Legyen a neve mondjuk SOROS, és az azonosítója S1. (A READY természetesen a gép válasza.)
Persze nemcsak új, hanem használt lemezt is újraformázhatunk, ha nem sajnáljuk elveszí-

teni a tartalmát. (Ilyenkor elég, ha csak a lemeznevet adjuk meg, mert akkor a NEW nem formázza újra a lemezt, csak a nyilvántartásokat törli. És persze a lemeznevet is átírja.)

```
OPEN 15,8,15

READY.
PRINT#15,"NEW:SOROS,S1"

READY.
CLOSE 15

READY.
```

Ellenőrizzük a formázás sikerességét! Töltsük be a lemez tartalomjegyzékét, és listázzuk ki a képernyőre!

```
LOAD"$",8

SEARCHING FOR $
LOADING
READY.

LIST

0 "SOROS" S1 2A
664 BLOCKS FREE.
READY
```

A sikertelen formázás eredményét alább látjuk.

```
LOAD"$",8

SEARCHING FOR $
?FILE NOT FOUND  ERROR
READY.
```

Minthogy adatállományokat még nem tudunk létrehozni, az állománykezelő lemezparancsokat a programkezelő parancsokkal próbáljuk ki. Ezt nyugodtan megtehetjük, mert mindkettő ugyanúgy működik.
Ezért most gépeljünk be egy tetszőleges rövid programot, és mentsük ki a lemezre.

```
NEW

READY.

10 FOR I=1 TO 9
20 PRINT I,I*I,I↑2
30 NEXT I
```

```
SAVE "MINTA",8

SAVING MINTA
READY.
```

Ellenőrizzük a kimentés helyességét!

```
LIST

10 FOR I=1 TO 9
20 PRINT I,I*I,I↑2
30 NEXT I

READY.

VERIFY "MINTA",8

SEARCHING FOR MINTA
VERIFYING
OK

READY.
```

Tegyünk ellenpróbát! Változtassuk meg a képernyőn levő programot, akár csak egyetlen karakterben is. Majd ellenőrizzük újra. Természetesen hibaüzenetet kell kapnunk.

```
LIST

10 FOR I=1 TO 8 :REM:<== MODOSITVA!
20 PRINT I,I*I,I↑2
30 NEXT I

READY.

VERIFY "MINTA",8

SEARCHING FOR MINTA
VERIFYING
?VERIFY  ERROR
READY.
```

Megjegyezzük, hogy a VERIFY sikertelen lehet más okból is, például ha a lemezen levő program megsérült, vagy ha a lemezegység meghibásodott stb. Feltételezzük, hogy esetünkben ennek a bekövetkezése valószínűtlen; ezért idéztük elő, a tárolt program megváltoztatásával, mesterségesen a hibaállapotot.

```
LIST

10 FOR I=1 TO 8 :REM:<== MODOSITVA!
20 PRINT I,I*I,I↑2
30 NEXT I

READY.

SAVE "MINTA",8

SAVING MINTA
READY.
```

Próbáljuk meg kimenteni az új programverziót a régi programnévvel! A kimentésnek nem szabad sikerülnie, hiszen a lemezen a programot a névnek egyértelműen kell azonosítania, így nem lehetnek azonos nevű programok.
Ilyenkor a képernyőn nem jelenik meg hibaüzenet, a sikertelen kimentést csak a lemezegység villogó piros lámpája jelzi.

FIGYELEM!
Ez az állományokra is vonatkozik. Sőt, még az sem megengedett, hogy egy állomány neve egy program nevével megegyezzen. Noha elvileg a lemezkezelő meg tudná különböztetni az állományt az azonos nevű programtól, ezt nem teszi meg.

Ha a kimentő parancsban jelezzük, hogy nem új kimentésről van szó, hanem egy meglevő program felülírásáról, akkor a kimentés sikeresen végrehajtódik.

```
LIST

10 FOR I=1 TO 8 :REM:<== MODOSITVA!
20 PRINT I,I*I,I↑2
30 NEXT I

READY.

SAVE "@:MINTA",8

SAVING @:MINTA
READY.
```

Adatállományokat nem lehet SAVE-vel lemezre vinni. Ott is fennáll viszont annak lehetősége, hogy az „@" jellel informálhatjuk a lemezkezelőt arról, hogy ne új állományt hozzon létre, hanem egy meglevőt írjon felül.
Megjegyezzük, hogy akár programok, akár állományok esetében ilyenkor nem tényleges felülírást hajt végre a lemezkezelő rendszer, hiszen lehetséges, hogy az új program vagy állomány el sem férne a régi helyén. Valójában a lemezkezelő kitörli a régi programot vagy

állományt, majd az újat elhelyezi valahol a lemezen, ott, ahol elegendő helyet talál számára. Természetesen a tartalomjegyzéket és a lemeztérképet ennek megfelelően módosítja.

FIGYELEM!

Programok felülírással való kimentésekor gyakran tapasztaltuk, hogy látszólag sikeres kimentés után, azaz amikor semmilyen hibaüzenetet nem kaptunk, és semmilyen hibajelenséget nem észleltünk, visszatöltéskor a kimentett program helyett mégis egy másik program jött be — egy olyan program, amely a lemezen egészen más névvel szerepelt.
E jelenséggel kapcsolatban az az álláspontunk, hogy amíg nem vagyunk meggyőződve a felülírás megbízhatóságáról, célszerű azt inkább elkerülni. Helyette előbb töröljük a programot, majd azután mentsük ki. Ezt az elvet ne csak programokra alkalmazzuk, hanem állományokra is terjesszük ki.

Ellenőrzésképpen töltsük be a lemezről a programot, és listázzuk ki a képernyőre! A módosított változatot kell megkapnunk.

```
LOAD "MINTA",8

SEARCHING FOR MINTA
LOADING
READY.
LIST

10 FOR I=1 TO 8 :REM:<== MODOSITVA!
20 PRINT I,I*I,I↑2
30 NEXT I
READY.
```

A programot a nyomtatóra is kilistázhatnánk, de ez most felesleges. Nem árt azonban, ha visszaemlékezünk erre a lehetőségre, mert később szükségünk lesz rá — nyilván csak az állománykezelő programoknál, maguknál az állományoknál nem, mivel azok így nem listázhatók.

```
OPEN 4,4

READY.
CMD4
LIST
PRINT#4

READY.
CLOSE 4

READY.
```

Készítsünk másolatot a lemezen levő programról! (Ha meg akarunk győződni a művelet sikerességéről, kérjük le, és listázzuk ki a lemez tartalomjegyzékét.)

```
OPEN 15,8,15

READY.
PRINT#15,"COPY:MASOLAT=MINTA"

READY.
CLOSE 15

READY.
```

Ez a másolatkészítési lehetőség soros adatállományokra is használható, pontosan ugyanebben a formában, sőt a COPY-val több soros állomány is összemásolható egyetlen állománnyá.

Ez az összemásolás nem hajtható végre sem programokra, sem nem-soros adatállományokra. Így most nem tudjuk kipróbálni. Ezért akkor fogunk részletesen visszatérni rá, ha majd a lemezünkön lesznek összekapcsolható soros állományok.
Változtassuk meg a program nevét, mondjuk a MASOLAT-ét FELESLEGES-re! Ilyenkor a lemezkezelő rendszer csupán a lemez tartalomjegyzékében írja át a program nevét a megadott új névre, így ez a művelet meglehetősen gyors.

```
OPEN 15,8,15

READY.
PRINT#15,"RENAME:FELESLEGES=MASOLAT"

READY.
CLOSE 15

READY.
```

Pontosan ugyanebben a formában használható adatállományok nevének megváltoztatására is, amire a soros feldolgozásra épülő rendszerekben nagyonis gyakran szükség lesz.
Töröljük a felesleges programot! Ugyanígy törölhetők a soros állományok is.

```
OPEN 15,8,15

READY.
PRINT#15,"SCRATCH:FELESLEGES"

READY.
CLOSE 15

READY.
```

Tudnunk kell azonban, hogy a program vagy állomány törlésekor a lemezkezelő csak a tartalomjegyzékben tárolt információkat törli, valamint szabadra állítja a lemeztérképen a

```
0 SOROS                    S1 2A
1      "MINTA"             PRG
41     "TARGYMUTATO"       SEQ
6      "HIBAJEGYZEK"       SEQ
0      "SIKERTELEN"       *SEQ
1      "PELDA"             PRG
605 BLOCKS FREE.

READY.
```

```
* LEMEZTERKEP *   * INTRONIK SOFTWARE *

    LEMEZ: SOROS                  S1

20:+++++++++++++●●●●------------------
19:+++++++++++++●++●------------------
18:+++++++++++++●●●●+++++++-----------
17:++++++++++++++++●+++++++++++++-----
16:+++++++++++++●●●●++++++++++++++++++
15:+++++++++++++++●●++++++++++++++++++
14:+++++++++++++●●●●++++++++++++++++++
13:+++++++++++++++●●++++++++++++++++++
12:+++++++++++++●+●●++++++++++++++++++
11:+++++++++++++++●●++++++++++++++++++
10:++++++++++++++●●●++++++++++++++++++
 9:+++++++++++++●++●++++++++++++++++++
 8:+++++++++++++●●●●++++++++++++++++++
 7:++++++++++++++++●++++++++++++++++++
 6:+++++++++++++●●●●++++++++++++++++++
 5:+++++++++++++++●●++++++++++++++++++
 4:+++++++++++++●●●●++++++++++++++++++
 3:+++++++++++++++●●++++++++++++++++++
 2:+++++++++++++●●+●++++++++++++++++++
 1:+++++++++++++++●●●+++++++++++++++++
 0:++++++++++++++●●●●●++++++++++++++++
            1         2         3
SV=12345678901234567890123456789012345
```

1.a) ábra. Tömörítés előtt

```
0 "SOROS                " S1 2A
1     "MINTA"               PRG
41    "TARGYMUTATO"         SEQ
6     "HIBAJEGYZEK"         SEQ
1     "PELDA"               PRG
615 BLOCKS FREE.

READY.
```

```
* LEMEZTERKEP *  * INTRONIK SOFTWARE *
--------------------------------------
   LEMEZ: SOROS                S1
---TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
20:+++++++++++++●●●●------------------
19:+++++++++++++●++●------------------
18:+++++++++++++●●●●+++++++-----------
17:++++++++++++++++●+++++++++++++-----
16:+++++++++++++●●●+++++++++++++++++++
15:+++++++++++++++●+++++++++++++++++++
14:+++++++++++++●●●+++++++++++++++++++
13:+++++++++++++++●+++++++++++++++++++
12:+++++++++++++●+●+++++++++++++++++++
11:+++++++++++++++●●++++++++++++++++++
10:++++++++++++++●●●++++++++++++++++++
 9:+++++++++++++●++●++++++++++++++++++
 8:+++++++++++++●●●●++++++++++++++++++
 7:+++++++++++++++++++++++++++++++++++
 6:+++++++++++++●●●●++++++++++++++++++
 5:+++++++++++++++●+++++++++++++++++++
 4:+++++++++++++●●●+++++++++++++++++++
 3:+++++++++++++++●+++++++++++++++++++
 2:+++++++++++++●●++++++++++++++++++++
 1:+++++++++++++++●●●+++++++++++++++++
 0:++++++++++++++●●●●●++++++++++++++++
---┴┴┴┴┴┴┴┴┴1┴┴┴┴┴┴┴┴┴2┴┴┴┴┴┴┴┴┴3┴┴┴┴┴
SV=12345678901234567890123456789012345
```

1.b) ábra. Tömörítés után

program vagy állomány számára lefoglalt területek jelzőit, de magát a programot vagy állományt fizikailag nem törli a lemezről. Ez persze nem jelenti azt, hogy a programhoz vagy állományhoz hozzáférhetünk, mert csak a nyilvántartásból tudhatnánk meg, hogy az hol van. Ezenkívül, bármilyen lemezreírás alkalmával a program vagy állomány területe, illetve tartalomjegyzéki bejegyzése bármikor felülíródhat, hiszen a lemezkezelő ezeket a helyeket szabadnak érzékeli, így írás ellen nem védi.
Tömörítsük a lemezt!

```
OPEN 15,8,15

READY.
PRINT#15,"VALIDATE"

READY.
CLOSE 15

READY.
```

Erre sok esetben nagyon is szükségünk van. Ha például egy soros állományt csak részben sikerül létrehoznunk, mert mondjuk a program valamilyen okból megszakadt, az adatokkal feltöltött területek foglaltak maradnak, és feleslegesen lemezterületet vesznek igénybe. Ezen az sem segít, ha az állományt töröljük a tartalomjegyzékből. A tömörítés azonban felszabadítja ezeket a hulladékterületeket. (Természetesen egyben törli is a csonka állomány bejegyzését a tartalomjegyzékből.)
Összehasonlításul közöljük egy lemezünk tömörítés előtti és tömörítés utáni tartalomjegyzékét és lemeztérképét. Jól megfigyelhető a lefoglalt blokkok száma közötti különbség — 1.a) és 1.b) ábra.

Úgynevezett hulladék blokkok persze nemcsak sikertelen létrehozáskor, hanem a programok vagy állományok felülírásakor is keletkezhetnek. A tömörítés (VALIDATE) hatására ezek a területek is hozzáférhetővé válnak.

FIGYELEM!
A random állományokkal sorozatunk egy későbbi kötetében fogunk részletesen foglalkozni. Azonban már most felhívjuk a figyelmet arra, hogy ezeknél a tömörítés nem használható, mivel azáltal, hogy a tömörítés felszabadítaná az állomány számára lefoglalt területeket (blokkokat), megsemmisítené magát az állományt.

Idézzünk elő hibaállapotot, például úgy, hogy kérjünk a lemezről tartalomjegyzéket, miközben a lemezegység ajtaja nyitva van. Hibajelzést kapunk.
Ha most az ajtót becsukjuk, és újra kérjük a tartalomjegyzéket, a parancsunk sikeresen végrehajtódik, és a hibaállapot is megszűnik. Ez általában más hibák esetén is így van.
Lehetnek azonban olyan hibák, amelyek következtében a lemezmeghajtó memóriájában tárolt tartalomjegyzék vagy lemeztérkép megsérül, felülíródik. Ilyenkor a hiba megszüntetése után is hibaállapotban marad a lemezegység — a működést jelző (piros) lámpája villog. Ebből az állapotából lehet az alábbi parancssorozattal kihozni. (Ha még ezzel sem, akkor a lemezegységet előbb ki, majd újra be kell kapcsolnunk.)

```
OPEN 15,8,15

READY.
PRINT#15,"INITIALIZE"

READY.
CLOSE 15

READY.
```

A LEMEZKEZELÉS SAJÁTOSSÁGAI

A COMMODORE lemezkezelése azért olyan nehézkes, mert csak a programok betöltésére (LOAD), kimentésére (SAVE) és ellenőrzésére (VERIFY) rendelkezik külön lemezkezelő parancsokkal.

Az összes többi lemezkezelő funkciót, úgymint a formázást, törlést stb. kerülő úton, de egységesen az OPEN, PRINT és CLOSE utasításokkal kell megoldanunk.

Ezek az utasítások azonban nem lemezkezelő, hanem *adatkezelő* utasítások. Az OPEN-nel adatállományokat lehet *megnyitni,* a CLOSE-zal *lezárni,* a PRINT-tel adatokat lehet egy *adatállományba felvinni.*

A lemezkezelő funkciók végrehajtásakor tehát úgy kell tennünk, mintha egy adatállományt akarnánk létrehozni, tehát azt először is szabályosan megnyitjuk:

```
OPEN 15,8,15
```

pontosan megadva az állomány azonosítóját, a lemezegység számát, majd az adatforgalom lebonyolítására kiválasztott csatornát.

Most már elárulhatjuk, hogy az OPEN utasítás első számadata, amelynek azután szerepelnie kell a PRINT és a hozzá tartozó CLOSE utasításban is, nem más, mint az *állomány azonosítója.* (Ez elvileg 1 és 255 között tetszőleges egész szám lehet, úgy hogy standard perifériák esetén 1—127, nem standard perifériákon pedig 128—255 számok használhatók.) A programban tehát az állományokat az OPEN-ben megadott egyedi sorszámokkal azonosítjuk. Mi már régen megállapodtunk abban, hogy amikor csak lehet, a csatorna számát fogjuk állományazonosítóként használni. Ezért szerepel az OPEN elején és végén is ugyanaz a szám.

Ezután össze kell szednünk a lemezkezelő funkció végrehajtásához szükséges információkat. Vegyük például a formázást. Ennél a szükséges információk: a funkciót magát meghatározó kulcsszó, a NEW; a lemeznév, mondjuk SOROS; és a lemezazonosító, mondjuk S1.

Ezekből az információkból, megfelelő *elhatároló jelek* felhasználásával, egyetlen karakteres adatot kell előállítanunk: "NEW:SOROS,S1". Mellesleg ennek az adatnak a hossza nem haladhatja meg a 40 karaktert, így például egy három állományt összemásoló COPY esetén igen könnyen zavarba jöhetünk, ha az állományok neve 9 karakteres.

Ha mindez megvan, ezt a szöveges adatot ki kell írnunk az OPEN utasítással megnyitott állományba:

```
PRINT#15,"NEW:SOROS,S1"
```

Amennyiben több parancsot nem kívánunk kiadni, az állományt, mint minden más állományt, a *használat befejezése után le kell zárnunk:*

```
CLOSE 15
```

Ugyanígy kell eljárnunk a többi lemezkezelő funkció esetén is, csak más és más információkból kell összeraknunk a kiírandó adatot.

De mi történik a kiírt adattal? Normális esetben az a kiválasztott adatcsatornán keresztül eljut a lemezkezelőhöz, „aki" azután gondoskodik arról, hogy az adat annak rendje és módja szerint felkerüljön a lemezre, éspedig annak az állománynak a lemezterületére, amelyikbe az adatot fel kívántuk vinni. Pontosan ez történne a lemezkezelő információkból összeállított adatunkkal is, ha nem éppen a 15-ös csatornára küldenénk ki.

A lemezkezelőnek ugyanis van egy kitüntetett csatornája, amelyet másképpen kezel, mint a többit. Ezen adatforgalom nem is bonyolítható le, ezért ezt a csatornát parancs (command) csatornának nevezzük. E csatorna azonosítója a COMMODORE gépen mindig 15.

Az adatcsatornák 2-től 14-ig számozhatók. Minthogy 16 csatorna van, ebből is látható, hogy a gép nem egyetlen kitüntetett csatornát kezel, hanem összesen hármat. Ezek közé tartozik a 0-ás és az 1-es csatorna, amelyekkel azonban ebben a könyvben nem foglalkozunk, minthogy nincs közvetlen szerepük az állományok kezelésében.

A lemezkezelő lesben áll a parancs-csatorna másik végén, és ha azon keresztül küldünk el valamilyen adatot, akkor azt nem viszi fel a lemezre, mint ahogyan azt a többi csatornáról érkező adattal tenné, hanem elemezni kezdi, és megpróbál értelmes információkat kihámozni belőle. Ha ez sikerül neki, akkor a kapott információknak megfelelően jár el. (A példánkban szereplő "NEW:SOROS,S1" adat alapján formázni fogja a lemezt, amit SOROS lemeznévvel és S1 azonosítóval fog ellátni.)

Amennyiben a beérkező adatot nem sikerül értelmeznie, vagy ha az információ nem egyértelmű, akkor hibaüzenetet küld vissza a parancs-csatornán, és természetesen nem hajt végre semmit.*

Az ily módon adatkezelésre visszavezetett lemezkezelés egyetlen előnye, hogy az *utasításai programból is kiadhatók,* anélkül, hogy ettől a BASIC fordító vagy értelmező programja bonyolultabbá válna. Hátránya viszont, hogy állandóan figyelemmel kell kísérnünk a parancs-csatorna állapotát. Hibaüzenetet kapunk ugyanis, ha megnyitatlan parancs-csatornára küldünk ki parancsot, de akkor is, ha a már megnyitott csatornát akarjuk megnyitni.

Természetesen a több állományt kezelő programokban nemcsak a parancs-csatornát, hanem az *egyes állományok adatcsatornáit* is meg kell nyitnunk. Ezekre ugyanazok a szabályok vonatkoznak, mint a parancs-csatornára. Írni, olvasni csak megfelelően nyitott állományba, illetve állományból lehet. Az újramegnyitás azonban itt is *tilos.* A lezáratlanul hagyott állományok pedig, amint később látni fogjuk, *megsemmisülnek.*

Bonyolítja a helyzetet, hogy bizonyos lemezműveletek automatikusan nyitják és zárják, mások pedig bizonyos szövegösszefüggésben automatikusan zárják a parancs-csatornát.

---

*Az „ALAPISMERETEK" kötetben a 81-ik oldalon már bemutattuk, hogy miként lehet a parancs-csatornáról leolvasni ezt a visszaküldött jelet. Erre a későbbiek során még ebben a kötetben is látunk majd példát.

Például, ha a már megnyitott csatornát újra megnyitjuk, FILE OPEN ERROR üzenetet kellene kapnunk, ez azonban nem mindig következik be. Ha a második megnyitásban az állomány azonosítója nem egyezik meg az első OPEN-ben megadott számmal, akkor nem jelenik meg hibaüzenet.

Ugyanez a helyzet, ha az első megnyitást parancsként, a másodikat pedig programból, utasításként adjuk ki, azaz a két megnyitás között RUN parancs hajtódik végre. Nem kapunk hibaüzenetet akkor sem, ha a két megnyitás között LOAD, SAVE vagy VERIFY parancsot hajtatunk végre.

Mindezek arra ösztönöznek, hogy valamilyen rendet tartsunk a parancs-csatorna kezelésében. A *módszeres nyitás* és *zárás* ugyanis megkímél bennünket attól, hogy számon kelljen tartanunk, mikor milyen feltételek bekövetkezése zavarja, illetve nem zavarja meg a rendszert, ha a csatornát nem olyan állapotban találja, mint amilyenben várja.

Fogadjuk el alapelvként, hogy csatornát feleslegesen ne tartsunk nyitva! Ez persze korántsem jelenti azt, hogy a csatornát minden egyes hozzányúlás alkalmával nyissuk és zárjuk. Megnyithatjuk, és mindaddig nyitva tarthatjuk, amíg adatforgalomra használjuk, de ha már nincs szükségünk rá, zárjuk le.

ÁLLOMÁNYKEZELŐ UTASÍTÁSOK

A soros állományok kezelésére az alábbi öt utasítás használható:

- OPEN : állomány megnyitása
- PRINT#: adat írása állományba
- INPUT#: adat olvasása állományból
- GET# : bájt olvasása állományból
- CLOSE : állomány lezárása

Használatukkal az állományok kezelése kapcsán részletesen fogunk foglalkozni. Itt elöljáróban csak az alapelveket rögzítjük le.

```
OPEN 2,8,2,"ALFA,SEQ,READ"
```

Ha az állományt már korábban megnyitottuk, a program FILE OPEN hibaüzenettel leáll. Egyébként megnyitja a 2-es állományt a 8-as egységen. Lefoglalja számára a 2-es adatcsatornát. Tudomásul veszi, hogy az állományt olvasni (READ) akarjuk. Ennek megfelelően megkeresi az "ALFA" nevű bejegyzést a lemez tartalomjegyzékében. Ha nem találja, vagy az nem soros (SEQ) állományé, akkor hibaüzenetet nem kapunk, a program tovább fut, de a lemezegység piros lámpája villog. Ilyenkor a parancs-csatornáról a FILE NOT FOUND üzenetet olvashatjuk le. Ha megtalálja, megjegyzi az adott állomány kezeléséhez ott tárolt adatokat (kezdőcím, méret stb.).

```
OPEN 3,8,3,"BETA,SEQ,WRITE"
```

Ha az állományt már korábban megnyitottuk, a program FILE OPEN hibaüzenettel leáll. Egyébként megnyitja a 3-as állományt a 8-as egységen. Lefoglalja számára a 3-as adatcsatornát. Tudomásul veszi, hogy az állományt létre akarjuk hozni, azaz írni (WRITE) akarunk bele. Ennek megfelelően megkeresi a "BETA" nevű bejegyzést a lemez tartalomjegyzékében. Ha megtalálja, hibaüzenetet nem kapunk, a program tovább fut, de a lemezegység piros lámpája villog. Ilyenkor a parancs-csatornáról FILE EXISTS üzenetet olvashatunk le. Ha

nem találja, akkor felviszi a tartalomjegyzékbe az állománynevet, és szabad helyet keres neki a lemezen.

```
OPEN 4,8,4,"GAMMA,SEQ,APPEND"
```

Ha az állományt már korábban megnyitottuk, a program FILE OPEN hibaüzenettel leáll. Egyébként megnyitja a 4-es állományt a 8-as egységen. Lefoglalja számára a 4-es adatcsatornát. Tudomásul veszi, hogy az állományt bővíteni (APPEND) akarjuk, azaz hozzá akarunk írni. Ennek megfelelően megkeresi a "GAMMA" nevű bejegyzést a lemez tartalomjegyzékében. Ha nem találja, hibaüzenetet nem kapunk, a program tovább fut, de a lemezegység piros lámpája villog. Ilyenkor a parancs-csatornáról FILE NOT FOUND üzenet olvasható le. Ha megtalálja, szabad helyet keres az állomány folytatásának a lemezen.

```
PRINT#5,"A";S$;1;S$;B
```

Ha az 5-ös állomány nincs megnyitva, a program FILE NOT OPEN üzenettel leáll. Ha az állomány olvasásra van megnyitva, hibaüzenetet nem kapunk, a program tovább fut, de a PRINT# utasítás nem hajtódik végre. Egyébként folyamatosan egymás után, sorban felviszi az 5-ös állomány lemezterületére, az első szabad helytől kezdve az "A" konstanst, az S$ változó tartalmát, az 1 konstanst, az S$ változó tartalmát, a B változó tartalmát, valamint egy sorvége (RETURN) jelet. Közben az adatok által elfoglalt blokkokat bejegyzi a lemeztérképre.

```
INPUT#6,X,Y$
```

Ha a 6-os állomány nincs megnyitva, a program FILE NOT OPEN üzenettel leáll. Ha az állomány nem olvasásra van megnyitva, hibaüzenetet nem kapunk, a program tovább fut, de az INPUT# utasítás nem hajtódik végre. Egyébként az utolsó beolvasást követő pozíciótól kezdve beolvas a 6-os állomány lemezterületéről két adatot; az elsőt az X változóba, a másodikat az Y$ változóba tölti be. Ha a betöltés sikertelen, például azért, mert az első adat nem numerikus, akkor a program FILE DATA ERROR üzenettel leáll. Egyébként az adatokat szeparátor(elválasztó)jeltől szeparátorjelig olvassa.
Később pontosan meg fogjuk mutatni, hogy mi számít szeparátorjelnek. Már most megjegyezhetjük azonban, hogy általában vagy a *vessző,* vagy a *sorvége* (RETURN) *jel.*

FIGYELEM!
Üres állományból, amit tehát a létrehozásakor csak megnyitottunk és lezártunk, *olvasni nem lehet.* Ilyenkor a *program lemerevedik,* és a gép csak STOP+RESTORE segítségével vagy ki/be kapcsolással hozható mozgásba.

```
GET#7,U$,V$,Z$
```

Ha a 7-es állomány nincs megnyitva, vagy nem olvasásra van megnyitva, ugyanez a jelenség következik be, mint az INPUT# esetén. Egyébként az utolsó olvasást követő pozíciótól kezdve beolvas a 7-es állomány lemezterületéről három egymást követő bájtot, és azokat sorban egymás után betölti az U$, V$, Z$ változókba. Nincs tekintettel arra, hogy a bájtok adatokhoz tartoznak-e, vagy adatok közötti szeparátorjeleket tartalmaznak.

FIGYELEM!
Üres állomány esetén a GET# is úgy viselkedik, mint az INPUT#.

```
CLOSE 8
```

Ha a 8-as állomány már le van zárva, hatástalan. Egyébként elengedi (felszabadítja) az állomány számára az OPEN-ben lefoglalt adatcsatornát. Ha az állomány nem olvasásra volt megnyitva, a tartalomjegyzékbe bejegyzi az állomány méretére és elhelyezkedésére vonatkozó információkat.

FIGYELEM!

A CLOSE ez utóbbi funkciójának a következménye, hogy ha egy írásra megnyitott állományt nem zárunk le, akkor az *állomány* tulajdonképpen *megsemmisül.* Még akkor is, ha a lemez tartalomjegyzékében megmarad a neve úgy, ahogyan azt az OPEN utasítás oda felírta. A jelenségre abból következtethetünk, hogy ilyenkor az állomány által elfoglalt blokkok száma a tartalomjegyzékben nullával van jelölve, és a SEQ jelzés előtt egy * (csillag) áll.

Megjegyezzük, hogy egy állomány felülírásra is megnyitható. Az ilyen megnyitás pontosan olyan, mint az írásra történő megnyitás, csak az állomány neve elé kell odabiggyeszteni a „kukac" jelet és egy kettőspontot (@:) — ugyanúgy, mint a programok felülírásakor. A működése azonban eltérő: ha az állományt már korábban megnyitottuk, FILE OPEN ERROR üzenettel a program leáll. Ha az állomány szerepel a tartalomjegyzékben, akkor felülíródik, ha nem, akkor ugyanúgy létrejön, mintha csak simán írásra nyitottuk volna meg.

A fenti állománykezelési lehetőségek néhány egyszerű tesztprogrammal könnyen kipróbálhatók.

Először hozzunk létre egy "PROBA" állományt:

```
10 OPEN 2,8,2,"PROBA,SEQ,WRITE"
20 PRINT#2,"AAAA"
30 CLOSE 2
99 END
```

Ennek tartalma egyetlen adat: "AAAA". Erről meggyőződhetünk, ha írunk egy programot, amely az állományt olvasni tudja:

```
10 OPEN 2,8,2,"PROBA,SEQ,READ"
20 INPUT#2,A$
25 PRINT A$
30 CLOSE 2
99 END
```

Most bővítsük az állományt egy újabb adattal:

```
10 OPEN 2,8,2,"PROBA,SEQ,APPEND"
20 PRINT#2,"BBBB"
30 CLOSE 2
99 END
```

A program végrehajtásakor az állomány bővülni fog a "BBBB" adattal. Ha lefuttatjuk az olvasóprogramot, persze csak az "AAAA" adatot kapjuk meg, mert ez a program csak egy adatot olvas. Könnyen írhatunk olyat, amelyik kettőt is tud:

```
10 OPEN 2,8,2,"PROBA,SEQ,READ"
20 INPUT#2,A$
21 INPUT#2,B$
25 PRINT A$
26 PRINT B$
30 CLOSE 2
99 END
```

. . .

Így már meggyőződhetünk arról, hogy az állományban mindkét adat szerepel. Végül írjuk felül az állományt:

```
10 OPEN 2,8,2,"@:PROBA,SEQ,WRITE"
20 PRINT#2,"CCCC"
30 CLOSE 2
99 END
```

. . .

Ha most végrehajtjuk bármelyik olvasóprogramot, láthatjuk, hogy az állományban csak egyetlen adat van, a "CCCC". A korábbi "AAAA" és "BBBB" adatok a felülírással megsemmisültek.

Még egy tanulság adódik, ha erre az állományra a két adatot olvasó programot futtatjuk le: az A$ tartalma "CCCC" lesz, a B$ viszont üres marad, vagyis a " " adatot tartalmazza. Egy állományból tehát nem olvasható több adat, mint amennyi van benne.

E mintaprogramocskák alkalmas módosításával, például az OPEN elhagyásával, hibás állománynév megadásával, az OPEN megduplázásával, a READ vagy WRITE kulcsszavak megcserélésével stb. előállíthatjuk a hibaállapotokat, és tanulmányozhatjuk azok következményeit. Ilyenkor célszerű a parancs-csatorna lekérdezését beépíteni a programokba. Például:

```
10 OPEN 15,8,15
20 OPEN 2,8,2,"PROBA,SEQ,WRITE"
30 INPUT#15,H,H$
31 PRINT : PRINT "<OPEN >";H;H$
40 PRINT#2,"AAAA"
50 INPUT#15,H,H$
51 PRINT : PRINT "<PRINT>";H;H$
60 CLOSE 2
70 INPUT#15,H,H$
71 PRINT : PRINT "<CLOSE>";H;H$
80 CLOSE 15
99 END
```

. . .

Ha ezt lefuttatjuk, hiba fog bekövetkezni, hiszen a "PROBA" állomány már létezik:

```
RUN

<OPEN > 63 FILE EXISTS
<PRINT> 0 OK
<CLOSE> 0 OK

READY.
```

Figyeljünk fel arra, hogy hibaállapot csak a megnyitásnál következett be. A többi műveletnél, azaz az írásnál és a lezárásnél hibajelzést nem kaptunk, viszont a műveletek hatástalanok voltak. (Ebben van bizonyos logika: nem lehet hibás az a művelet, amely nem hajtódik végre.)

ADATKEZELÉS SOROS ÁLLOMÁNYOKBAN

A lemezről leolvasni csak azt lehet, ami oda fel volt írva. Ezért az írás és az olvasás között természetszerűleg összhangnak kell lennie. Ahhoz, hogy ezt megértsük, nézzük meg előbb, hogy milyen formában vannak az adatok a lemezen tárolva.

Mindenekelőtt írjunk egy tesztprogramot, amely bármely soros állományt bájtonként tud olvasni, vagyis nem INPUT, hanem GET utasítást használ. A GET ugyanis nem „érzékeli" az adatokat, az állományból csak bájtokat „lát"

```
1 PRINT
2 PRINT
3 PRINT"ALLOMANY ADATONKENTI OLVASASA"
4 PRINT"-----------------------------"
5 PRINT
100 REM:
101 REM: --------------------------
102 REM:
110 :   INPUT "ALLOMANY=";F$
120 :   I=1
130 :   OPEN 2,8,2,F$+",SEQ,READ"
140 :   PRINT:PRINT
200 REM:
201 REM: --------------------------
202 REM:
210 :   IF STATUS THEN GOTO 310
220 :   :   INPUT#2,A$
230 :   :   PRINT RIGHT$("  "+STR$(I),2);
240 :   :   PRINT ". ADAT=";A$;"♠"
250 :   :   PRINT:I=I+1
299 :   GOTO 210
300 REM:
301 REM: --------------------------
```

```
302 REM:
310 :   PRINT "[VEGE]"
320 :   CLOSE 2
330 :   PRINT
900 REM:
901 REM: ------------------------------------------------
902 REM:
999 END

READY.
```

A program bejelentkezik (1–4), majd bekéri az olvasandó állomány nevét és megnyitja azt olvasásra (110–140). Ezután megszervez egy ciklust (210–299), amelyben beolvas egy bájtot (220), ha az sorvége (RETURN) jel, akkor kicseréli egy kis rombuszra (230), majd kinyomtatja a bájt tartalmát a képernyőre (240). Minden olvasás előtt figyeli az állomány végét (210), és ha elérte azt, megfelelő üzenetet ír ki, és lezárja az állományt (310–330). Végül leáll (999).

Megjegyezzük, hogy az állomány végének figyelése rendkívül egyszerű. A lemezkezelő ugyanis automatikusan beállít egy STATUS nevű változót, amely semmilyen más célra nem használható. Ennek tartalma logikai „hamis", ha még nem érte el az állomány végét, és logikai „igaz", ha már elérte.

Valójában a STATUS értéke az állomány végének elérése előtt 0, az állomány végének elérésekor 64, sikertelen lemezművelet esetén pedig 128. Ezeket a numerikus értékeket használjuk logikai értékként. (Érdemes megfigyelni, hogy mind a 64, mind pedig a 128 esetén csak egy bit értéke = 1.)

Már most felhívjuk a figyelmet arra, hogy a STATUS értéke nem az utolsó adat beolvasása után vált át 0-ról 64-re, hanem *már az utolsó adat beolvasásakor.* További sajátossága, hogy csak egyetlen STATUS változó van, amely mindig a legutolsó művelet sikerességét jelzi. Ez bizony rendkívül kellemetlen, ha több állományt kell kezelnünk, de egyetlen állomány esetén, mint a példánkban is, az állomány végének figyelésére ez a legegyszerűbb módszer.

FIGYELEM!

Az üres állományok olvasása ellen a STATUS nem jelent védelmet. Értéke a megnyitáskor még nulla, és csak az első olvasáskor áll át 64-re. De akkor már késő.

Csupán a teljesség kedvéért megemlítjük, hogy a soros állományok végén *nincs külön állományvége* (EOF = end-of-file) jel. Azt, hogy elfogytak az adatok, a lemezkezelő a blokk feltöltöttségéből tudja meg. Ezt az adatot magában a blokkban tárolja, az első két bájt közül a másodikon. Ezért egy blokkba legfeljebb 254 bájtnyi adat írható. A STATUS és az állományvége kezelésével egyébként a későbbiekben még foglalkozni fogunk.

Most egyelőre ott tartunk, hogy kipróbálhatjuk a bájtonként olvasó programunkat a "PROBA" állományra. Ha mindeddig a leírtak szerint jártunk el, akkor az eredmény ez lesz:

```
RUN

ALLOMANY BAJTONKENTI OLVASASA
-----------------------------

ALLOMANY=? PROBA

CCCC♦[VEGE]

READY.
```

A sorvége (RETURN) jelet azért cseréltük ki a kis rombuszra (káró jelre), hogy a kiírás olvashatóbb legyen.
Ennek mintájára könnyen írhatunk olyan programot is, amellyel soros állományokat adatonként tudunk olvasni:

```
1 PRINT
2 PRINT
3 PRINT"ALLOMANY ADATONKENTI OLVASASA"
4 PRINT"-----------------------------"
5 PRINT
100 REM:
101 REM: -------------------------
102 REM:
110 :   INPUT "ALLOMANY=";F$
120 :   I=1
130 :   OPEN 2,8,2,F$+",SEQ,READ"
140 :   PRINT:PRINT
200 REM:
201 REM: -------------------------
202 REM:
210 :   IF STATUS THEN GOTO 310
220 :   :   INPUT#2,A$
230 :   :   PRINT RIGHT$("  "+STR$(I),2);
240 :   :   PRINT ". ADAT=";A$;"♦"
250 :   :   PRINT:I=I+1
299 :   GOTO 210
300 REM:
301 REM: -------------------------
302 REM:
310 :   PRINT "[VEGE]"
320 :   CLOSE 2
330 :   PRINT
900 REM:
901 REM: -------------------------
902 REM:
999 END

READY.
```

Az eltérés csak annyi, hogy GET# helyett INPUT# utasítást használtunk, így 1-1 bájt helyett 1-1 adat fog beolvasódni a ciklus minden egyes menetében. Természetesen a jobb olvashatóság kedvéért a kiírás formátuma is más.
Futtassuk le ezt a programot is a "PROBA" állományra:

```
RUN

ALLOMANY ADATONKENTI OLVASASA
-----------------------------

ALLOMANY=? PROBA

 1. ADAT=CCCC
[VEGE]

READY.
```

Végül hozzunk létre egy üres állományt:

```
10 OPEN 2,8,2,"URES,SEQ,WRITE"
20 CLOSE 2

READY.
```

Majd engedjük rá a bájtonkénti olvasóprogramot. Amint vártuk, a rendszer el fog szállni. Ne ijedjünk meg, ki/be kapcsolással vagy a STOP—RESTORE használatával helyreállítható.
Érdekes módon a probléma mégis megkerülhető. Igaz ugyan, hogy a STATUS az OPEN után nem vált át, de ha az utóbbit a parancs-csatorna lekérdezése követi, akkor mégis átáll 64-re, ha az állomány a megnyitáskor üres volt.
Ezt egy kis tesztprogrammal könnyen kipróbálhatjuk:

```
110 PRINT
120 OPEN 15,8,15
130 OPEN 2,8,2,"URES,SEQ,READ"
140 PRINT "OPEN      UTAN STATUS =";STATUS
150 PRINT
160 INPUT#15,H,H$
170 PRINT "INPUT#15 UTAN STATUS =";STATUS
180 CLOSE 2
190 CLOSE 15
199 END

RUN
OPEN      UTAN STATUS = 0
INPUT#15 UTAN STATUS = 64
READY.
```

Ha ennek értelmében beszúrjuk az alábbi utasításokat a bájtonként olvasó programba:

```
125 OPEN 15,8,15
135 INPUT#15,H,H$
325 CLOSE 15
```

akkor újra megkísérelhetjük az "URES" állomány olvasását. Most már kielégítő eredményt kapunk:

```
RUN

ALLOMANY BAJTONKENTI OLVASASA
-----------------------------

ALLOMANY? URES

[VEGE]

READY.
```

Látható, hogy a program nem szállt el, hanem az olvasó ciklus egyszer sem hajtódott végre.

Mint már említettük, a STATUS használata az állomány végének figyelésére a legjobb eszköz, különösen, ha egyetlen állományt kezel a programunk. Ezért célszerű, hogy éljünk is ezzel a lehetőséggel, azaz az OPEN után mindig kérdezzük le a parancs-csatornát. (Ez az olvasások után már nem feltétlenül szükséges.) Így igen tiszta, világos, logikus programszerkezeteket lehet létrehozni, aminek a rekordonkénti soros feldolgozásnál látjuk majd hasznát. Ugyanis ilyenkor a feldolgozó ciklus pontosan annyiszor fog végrehajtódni, ahány rekord van az állományban; mégpedig az első rekordot az első ciklusmenet, a másodikat a második, és így tovább, dolgozza fel. Természetes, hogy az üres állomány esetén, amelyben nulla számú rekord van, a feldolgozó ciklus nullaszor, azaz egyszer sem hajtódik végre. Az ilyen programok azon túlmenően, hogy jól áttekinthető szerkezettel rendelkeznek, az üres állományokkal is a rendeltetésüknek megfelelően működnek, tehát az állomány feltöltöttségétől függetlenül használhatók.

Így felvértezve, most már hozzáfoghatunk az írás/olvasás teszteléséhez. Vegyük észre, hogy a bájtonként és az adatonként olvasó programok nem gyakorló, hanem általánosan használható, igazi tesztprogramok. Ezért szerepelnek itt a végleges dokumentációjuknak megfelelő formában.

Mindenekelőtt hozzunk létre egy állományt, amely négy numerikus adatot tartalmaz úgy, hogy minden egyes adatot külön-külön vigyünk fel:

```
10 PRINT:PRINT " NUMERIKUS FELVITEL"
20 OPEN 2,8,2,"NUMPROBA,SEQ,WRITE"
30 PRINT#2,1111
40 PRINT#2,22.22
50 PRINT#2,-0.33
60 PRINT#2,+4000000000
```

```
70 CLOSE 2
80 PRINT:PRINT " KESZ"
99 END

READY.
```

A programot lefuttatva létrejön a "NUMPROBA" állomány. Olvassuk ezt el a bájtonként olvasó programmal:

```
RUN

ALLOMANY BAJTONKENTI OLVASASA
-----------------------------

ALLOMANY=? NUMPROBA

 1111 ◆ 22.22 ◆-.33 ◆ 4E+09 ◆[VEGE]

READY.
```

Láthatjuk egyrészt, hogy a numerikus adatok olyan formában kerülnek a lemezre, mint ahogyan a képernyőre kerülnének a PRINT hatására. Vagyis előjelbájttal, az adatot követő elválasztó szóközzel, valamint a túl nagy vagy túl kicsi számok esetén normál alakra konvertálva.

Másrészt a külön-külön felvitt adatok közé sorvége (RETURN) jelek kerülnek.

Olvassuk el az állományt az adatonként olvasó programmal:

```
RUN

ALLOMANY ADATONKENTI OLVASASA
-----------------------------

ALLOMANY=? NUMPROBA

 1. ADAT=1111 ◆
 2. ADAT=22.22 ◆
 3. ADAT=-.33 ◆
 4. ADAT=4E+09 ◆
[VEGE]

READY.
```

Két tanulságot állapíthatunk meg. Egyrészt az adatonként feltöltött állomány adatonként is olvasható; másrészt a numerikus adatok karakteres változókba is beolvashatók.

A numerikus olvasást ugyanezzel az állománnyal és ugyanezzel a programmal próbálhatjuk ki, ha az olvasóprogram A$ változóját mindkét előfordulási helyén A-ra cseréljük ki.

A karakteres írás/olvasás jellegzetességeivel a "PROBA" állomány kezelése kapcsán már

megismerkedtünk. Az alapelvek változatlanok: a karakteres adat olyan formában kerül a lemezre, mint amilyenben kiíráskor a képernyőn megjelenne; a külön-külön felvitt adatokat egy-egy sorvége (RETURN) jel követi; az így létrehozott állomány adatonként olvasható — természetesen csak a felvitel sorrendjében.
Bájtonként minden (nem üres) soros állomány olvasható.

REKORDOK KEZELÉSE SOROS ÁLLOMÁNYON BELÜL

Az adatok nemcsak külön-külön, hanem úgynevezett rekordokba (logikailag összetartozó adatcsoportokba) tömörítve is felvihetők a soros állományba. A rekordonkénti felvitel épülhet fix vagy változó hosszúságú rekordokra.

A *fix hosszúságú rekordokra* jellemző, hogy meghatározott számú karakterből álló folyamatos jelsorozatból állnak, amelyen belül az adatok semmilyen szeparátorjellel nincsenek elválasztva egymástól. A rekordot mindig sorvége (RETURN) jel zárja le.
Hogy ez a jelsorozat visszaolvasáskor értelmezhető, azaz adatokra szétbontható legyen, minden adatnak a rekord meghatározott pozícióján kell elhelyezkednie.

Az egyik megoldás, hogy a rekordot az adatokból összeszerkesztjük, majd egyetlen karaktersorozatban felvisszük az állományba. Legyen például két adatunk, egy 8 bájtos karakteres (szöveges), és egy 6 bájtos numerikus:

```
10 PRINT
20 PRINT"FIX REKORD FELVITEL"
30 PRINT
40 U$="              "
50 A$="AAAA" : B=1111
60 OPEN 2,8,2,"FIXREKORD,SEQ,WRITE"
70 X$=LEFT$(A$+U$,8)+RIGHT$(U$+STR$(B),6)
80 PRINT#2,X$
90 CLOSE 2
99 END

READY.
```

Minthogy az adatok nem feltétlenül a meghatározott hosszal rendelkeznek, gondoskodnunk kell a kiegészítésükről. Ez történik a 70-es sorban. A karakteres adatokat konvenció szerint balra, a numerikusakat jobbra zárjuk.
Olvassuk el az állományt a bájtonkénti olvasóval:

```
RUN

ALLOMANY BAJTONKENTI OLVASASA
-----------------------------------------------------------

ALLOMANY=? FIXREKORD

AAAA        1111◆[VEGE]

READY.
```

Látható a rekord, az összefüggő karaktersorozattal. Az adatonkénti olvasóval nem érdemes az állományt olvasni, mert a teljes rekord jönne be. (Próbáljuk ki!) Az egyes adatokhoz csak egy rekordonként olvasó programmal férhetünk hozzá:

```
10 PRINT
20 PRINT "FIX REKORD OLVASAS"
30 PRINT
40 OPEN 2,8,2,"FIXREKORD,SEQ,READ"
50 INPUT#2,X$
60 CLOSE 2
70 A$=LEFT$(X$,8)
75 PRINT "A$=<";A$;">"
80 B =VAL(RIGHT$(X$,6))
85 PRINT "B =<";B ;">"
90 PRINT
99 END

READY.
```

A program a teljes rekordot olvassa, majd a karakterláncot szétszedve előállítja az adatokat. Ehhez természetesen ismernie kell a rekord szerkezetét: az adatok sorrendjét, hosszát, típusát.

Ennek a felviteli módnak a hátránya, hogy az adatok összeszerkesztésével legfeljebb 255 bájtos rekordot hozhatunk létre, mivel ennél hosszabb karakterlánc nem állítható elő. Ugyanakkor az összeszerkesztés időigényes is, ami a programot lassítja.

Az összeszerkesztés azonban elkerülhető:

```
10 PRINT
20 PRINT"FIX REKORD FELVITEL"
30 PRINT
40 U$="              "
50 A$="AAAA" : B=1111
60 OPEN 2,8,2,"FIXREKORD,SEQ,WRITE"
70 A$=LEFT$(A$+U$,8)
75 B$=RIGHT$(U$+STR$(B),6)
80 PRINT#2,A$;B$
90 CLOSE 2
99 END

READY.
```

Itt a rekordot nem állítjuk össze a memóriában, hanem a PRINT# utasításban felsoroljuk a rekord adatait meghatározó konstansokat és változókat – egymástól pontosvesszővel elválasztva. Ennek hatására az adatok sorra felíródnak a lemezre, de közéjük nem kerül szeparátorjel – ugyanúgy mint a képernyőn. Így ez a program pontosan ugyanazt a rekordot állítja elő, mint az előző.

(Ha ki akarjuk próbálni, előbb a "FIXREKORD" állományt törölnünk kell.)

FIGYELEM!
Soha *ne használjunk* a változók között *a pontosvessző helyett vesszőt.* Ilyenkor az adatok ugyanúgy tabulálva kerülnek a lemezre, mint ahogyan a vessző hatására kiíratáskor (PRINT-tel) a képernyőre vagy a nyomtatóra kerülnének. Ez pedig jelentős helypocsékolást eredményez. Nem érdemes kipróbálnunk, de az eredményt bemutatjuk:

```
10 PRINT
20 PRINT"FIX REKORD FELVITEL"
30 PRINT
40 U$="      "
50 A$="AAAA" : B=1111
60 OPEN 2,8,2,"FIXREKORD,SEQ,WRITE"
70 A$=LEFT$(A$+U$,8)
75 B$=RIGHT$(U$+STR$(B),6)
80 PRINT#2,A$,B$
90 CLOSE 2
99 END

READY.

RUN

ALLOMANY BAJTONKENTI OLVASASA
-----------------------------

ALLOMANY=? FIXREKORD

AAAA                    1111♦[VEGE]

READY.
```

Megjegyezzük, hogy akár adatonként is felvihetnénk a rekord elemeit, azaz minden változót külön PRINT# utasításba írva, ha az utolsó adat PRINT#-je kivételével minden PRINT#-et pontosvesszővel fejezünk be. Ezt a megoldást azonban nem javasoljuk, mert kevésbé áttekinthető, a dokumentatív ereje is kisebb, és főleg a hibalehetőség nagyobb. (Ha például egy helyen kifelejtjük a pontosvesszőt, egy helyett két rekord kerül a lemezre — persze az egyik az eredeti rekord első felét, a másik a második felét tartalmazza.)

FIGYELEM!
Az INPUT# utasítással legfeljebb 88 karakter hosszúságú adatok olvashatók le a lemezről, így az ilyen fix hosszúságú rekordok mérete legfeljebb 88 bájt lehet.

A *változó hosszúságú rekordokban* minden adat pontosan annyi helyet foglal el, amennyi a tárolásához szükséges. Így nincs szükségünk sem az összeszerkesztésre, sem a kiegészítésre, de még a szétbontásra sem.

Ennek az az ára, hogy az adatok közé alkalmas *elválasztó jeleket kell elhelyeznünk,* hiszen (szemben a fix hosszúságú rekorddal) soha nem tudhatjuk, hogy hol végződik egy adat, és

hol kezdődik a következő. Elvileg többféle szeparátorjel használható lenne, de gyakorlatilag célszerű olyant választani, amely a rekord beolvasását megkönnyíti. Ez pedig a vessző.

Hozzunk létre egy változó hosszúságú rekordot:

```
10 PRINT
20 PRINT "VALTOZO REKORD FELVITEL"
30 PRINT
40 A$="BBBB" : B=2222
50 V$=","
60 OPEN 2,8,2,"VALREKORD,SEQ,WRITE"
70 PRINT#2,A$;V$;B
80 CLOSE 2
99 END

READY.
```

A felírást ugyanolyan PRINT# utasítás hajtja végre, mint amilyet a fix hosszúságú rekordnál használtunk. A különbség az, hogy az egyes adatok között még a szeparátorjelet is fel kell vinni. Fontos, hogy a felsorolt változókat *pontosvesszővel* válasszuk el egymástól, mert így biztosítható, hogy az adatok közé a szeparátorjelen kívül más ne kerüljön.

Leolvasva a rekordot a bájtonkénti olvasóval, láthatjuk, hogy a rekord adatai között ott a vessző, és a rekord végén a RETURN jel:

```
RUN

ALLOMANY BAJTONKENTI OLVASASA
-----------------------------

ALLOMANY=? VALREKORD

BBBB, 2222 ♦[VEGE]

READY.
```

Ez a rekord azonban az adatonkénti olvasóval nem olvasható be. Próbáljuk ki:

```
RUN

ALLOMANY ADATONKENTI OLVASASA
-----------------------------

ALLOMANY=? VALREKORD

 1. ADAT=BBBB♦
[VEGE]

READY.
```

A rekord beolvasásához rekordonként olvasó programot kell írnunk:

```
10 PRINT
20 PRINT "REKORDONKENTI OLVASAS"
30 PRINT "---------------------"
40 PRINT
50 OPEN 2,8,2,"VALREKORD,SEQ,READ"
60 INPUT#2,A$,B
70 CLOSE 2
80 PRINT:PRINT "A$=<";A$;">"
90 PRINT:PRINT "B =<";B ;">"
99 END

READY.
```

Ennek jellegzetessége, hogy az egy rekordból beolvasandó adatokat fogadó összes változó egyetlen INPUT# utasításban van. Természetesen az adatok sorrendjét és típusát ismernünk kell (de a hosszát nem!).

```
RUN
REKORDONKENTI OLVASAS
---------------------

A$=<BBBB>
B =< 2222 >
READY.
```

Itt tehát valóban rekordonkénti feldolgozás valósul meg. Minden PRINT# vagy INPUT# mindig egy teljes rekordot visz fel, illetve olvas be.

Megjegyezzük, hogy az INPUT#-ban megadott változók számának nem kell megegyeznie a rekordban levő adatok számával. Ha csak az A$ változót adjuk meg, akkor csak a rekord első adata fog betöltődni. Ha pedig egy harmadik, mondjuk C$ változót is megadunk, az nem fog értéket kapni. A rekordonként olvasó program megfelelő módosításával ezt könnyen kipróbálhatjuk. Itt csak a második esetre vonatkozó eredményt közöljük:

```
RUN
REKORDONKENTI OLVASAS
---------------------

A$=<BBBB>
B =< 2222 >
C$=<>
READY.
```

Fontos, hogy adatot átlépni nem lehet. Ha egy rekordban három adat van, de az INPUT#-ban csak két változót adunk meg, mindig az első két adat fog betöltődni, soha nem az első és az utolsó, vagy az utolsó kettő.

Ha pedig az INPUT#-ból úgy hagyunk el változót, hogy az őt követő vessző megmarad, azaz az INPUT# vesszővel kezdődik vagy vesszővel végződik, vagy két változó között több vessző van, akkor SYNTAX ERROR (helyesírási hiba) üzenetet kapunk. Ilyenkor a program futása megszakad.

A rekordonként létrehozott állomány csak a rekordszerkezet ismeretében kezelhető. Az adatok sorrendjét és típusát mindenképpen számon kell tartanunk — ez persze igaz az adatonként létrehozott állományra is. Fix hosszúságú rekordok esetén pedig ezenkívül még az adatok hosszát is ismernünk kell.

A sorrend határozza meg ugyanis az adat jelentését. Ha beolvasunk három adatot, legyenek ezek mondjuk 01, 02, 03, tudnunk kell, hogy melyik jelenti az évet, a hónapot és a napot. Nem mindegy, hogy ezek 1901. február 3-át, vagy 1903. január 2-át jelentik.

A típus ismerete azért fontos, mert numerikus változóba nem olvashatunk be szöveges (karakteres) adatot.

Próbáljuk meg kicserélni a rekordonként olvasó programban az A$ változót A-ra! Programmegszakítás és adathibára utaló üzenet lesz az eredménye:

```
RUN
REKORDONKENTI OLVASAS
---------------------

?FILE DATA  ERROR IN 60
READY.
CLOSE 2

READY.
```

A típusra természetesen nemcsak a rekordonkénti állománykezelésnél kell tekintettel lennünk, hanem bármilyen más olvasásnál is.

Megjegyezzük, hogy ez elkerülhető lenne, ha a lemezről mindig csak karakteres változókba olvasnánk be adatot. Amint már láttuk, ez numerikus adattal is megtehető. Ilyenkor azonban azokat az adatokat, amelyeket numerikusként kell használnunk, beolvasás után külön konvertálnunk kellene. Ez lehetséges, de felesleges időveszteséggel jár. Nyilvánvaló azonban, hogy ismeretlen állomány esetén, például a mi olvasó tesztprogramjainkban, nincs más választásunk.

Végezetül megemlítjük még a vegyes adatfelvitel lehetőségét. Ez úgy érhető el, hogy szeparátorjelként nem a vesszőt, hanem a sorvége (RETURN) jelet használjuk:

```
10 PRINT
20 PRINT "VALTOZO REKORD FELVITEL"
30 PRINT
40 A$="CCCC" : B=3333
50 V$=CHR$(13)
60 OPEN 2,8,2,"VALREKORD,SEQ,WRITE"
70 PRINT#2,A$;V$;B
80 CLOSE 2
99 END

READY.
```

Ilyenkor a felvitel történhet akár adatonként, akár rekordonként.
Miután töröltük a "VALREKORD" állományt, hozzuk létre ismét, de most a fenti programmal, rekordonként. Majd olvassuk el a bájtonkénti olvasóval:

```
RUN

ALLOMANY BAJTONKENTI OLVASASA
-----------------------------

ALLOMANY=? VALREKORD

CCCC◆ 3333 ◆[VEGE]

READY.
```

Látható, hogy minden adat után sorvége jel van. Kérdéses, hogy mit nevezhetünk itt rekordnak? Eddig az adatokat szeparátorjel (vessző), a rekordokat sorvégjel (RETURN) választotta el egymástól.
Nem célszerű ezt az elvet közvetlenül alkalmazni a mostani esetünkre. Hiszen akkor vagy azt kellene mondanunk, hogy az állományban nincsenek rekordok, vagy azt, hogy az állományban minden adat önálló rekord.
Gyakorlati megfontolásokból javasoljuk, hogy tekintsük az összetartozó adatok sorozatát, a példánkban a "CCCC" és 3333 értékeket egy *logikai rekordnak.*
Ilyen értelemben ez az állomány rekordonként is olvasható:

```
RUN
REKORDONKENTI OLVASAS
---------------------

A$=<CCCC>
B =< 3333 >
READY.
```

Ugyanakkor adatonként is végrehajtható az olvasás:

```
RUN

ALLOMANY ADATONKENTI OLVASASA
-----------------------------

ALLOMANY=? VALREKORD

 1. ADAT=CCCC◆
 2. ADAT=3333 ◆
[VEGE]

READY.
```

Ily módon ez a vegyes jellegű állomány adatonként és változó hosszúságú (logikai) rekordonként egyaránt kezelhető. Használata – a fenti, talán kissé akadémikusnak tűnő probléma

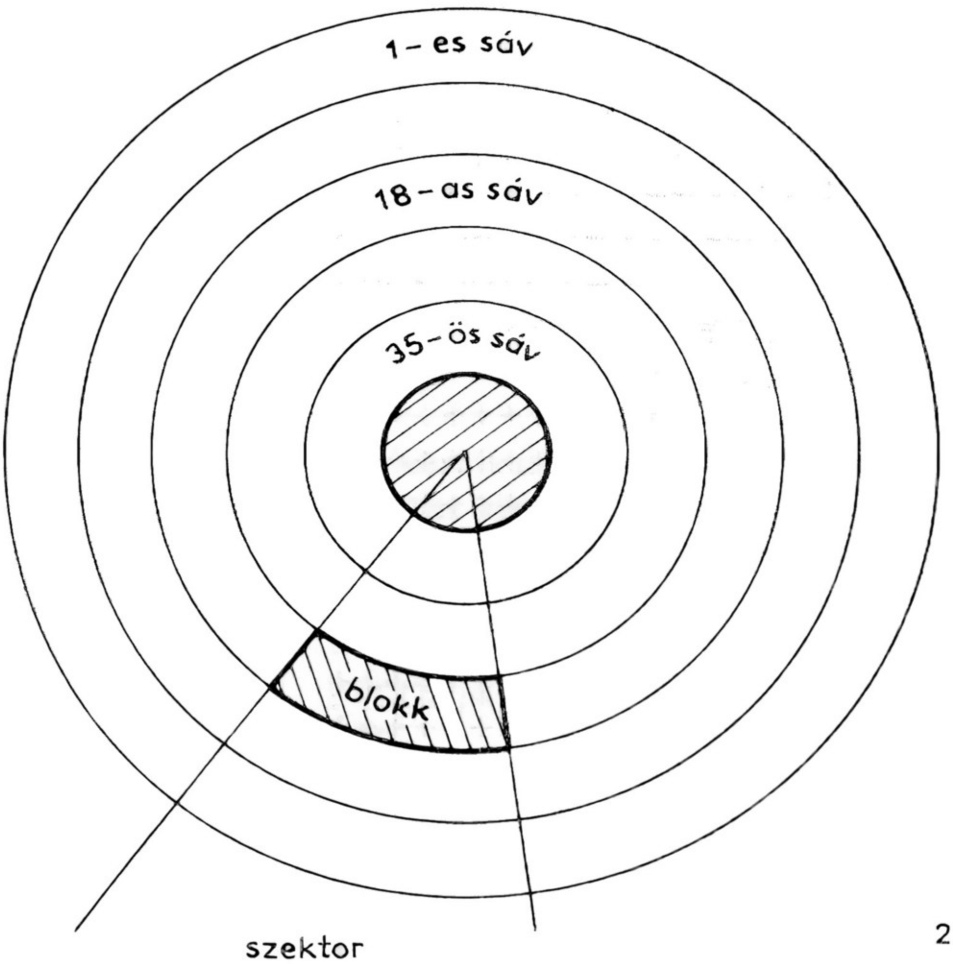
ellenére – elfogadható, azokban az extrém esetekben, amikor mindkét kezelési formára szükségünk lehet.
A második részben bemutatandó gyakorló programjaink nagyrészt ilyenek lesznek, pusztán azon egyszerű technikai okból, hogy a kipróbálásuk, tesztelésük során az állományok adatait az adatonként olvasó programmal könnyen ellenőrizhessük.

A LEMEZKEZELÉSI SZABÁLYOK ÖSSZEFOGLALÁSA

- Minden állományt használat előtt meg kell nyitni.
- Megnyitott állományt újra megnyitni nem szabad.
- Olvasásra megnyitni csak a létező állományt lehet.
- Írásra megnyitni csak még nem létező állományt lehet.
- Továbbírásra megnyitni csak létező állományt lehet.
- Felülírásra megnyitni akár létező, akár nemlétező állományt lehet.
- Írni csak nem olvasásra megnyitott állományba lehet.
- Olvasni csak olvasásra megnyitott állományból lehet.
- Az állományba bármilyen típusú adat felvihető.
- Az állományból csak olyan adat olvasható, amilyent felírtunk.
- Az állományból csak annyi adat olvasható, amennyit felírtunk.
- Numerikus változóba csak numerikus adat olvasható be.
- Karakteres változóba bármilyen típusú adat beolvasható.
- Használat után minden állományt le kell zárni.
- Lezárt állomány újra lezárható.
- Másolni csak lezárt állományt lehet.
- A parancs-csatornát a vele párhuzamosan nyitott adatcsatornák megnyitása előtt kell megnyitni.
- A parancs-csatornát a vele párhuzamosan nyitott adatcsatornák lezárása után lehet lezárni.

A lemez fizikai sajátosságai

Ahhoz, hogy kihasználhassuk a COMMODORE lemezkezelési lehetőségeit, ismernünk kell a lemez néhány alapvető tulajdonságát.
Ezek közül az egyik legfontosabb a lemez szerkezete:



2. ábra. A mágneslemez szerkezete

A lemez területe *sávokra* van felosztva. Összesen 35 sáv van egy lemezoldalon. Ezeket 1-től 35-ig terjedő számozással, úgynevezett *sávcímmel* azonosítjuk.
Az egyes sávok *szektorokból* állnak. A hosszabb külső sávokon több, a rövidebb belső sávokon kevesebb szektor van.

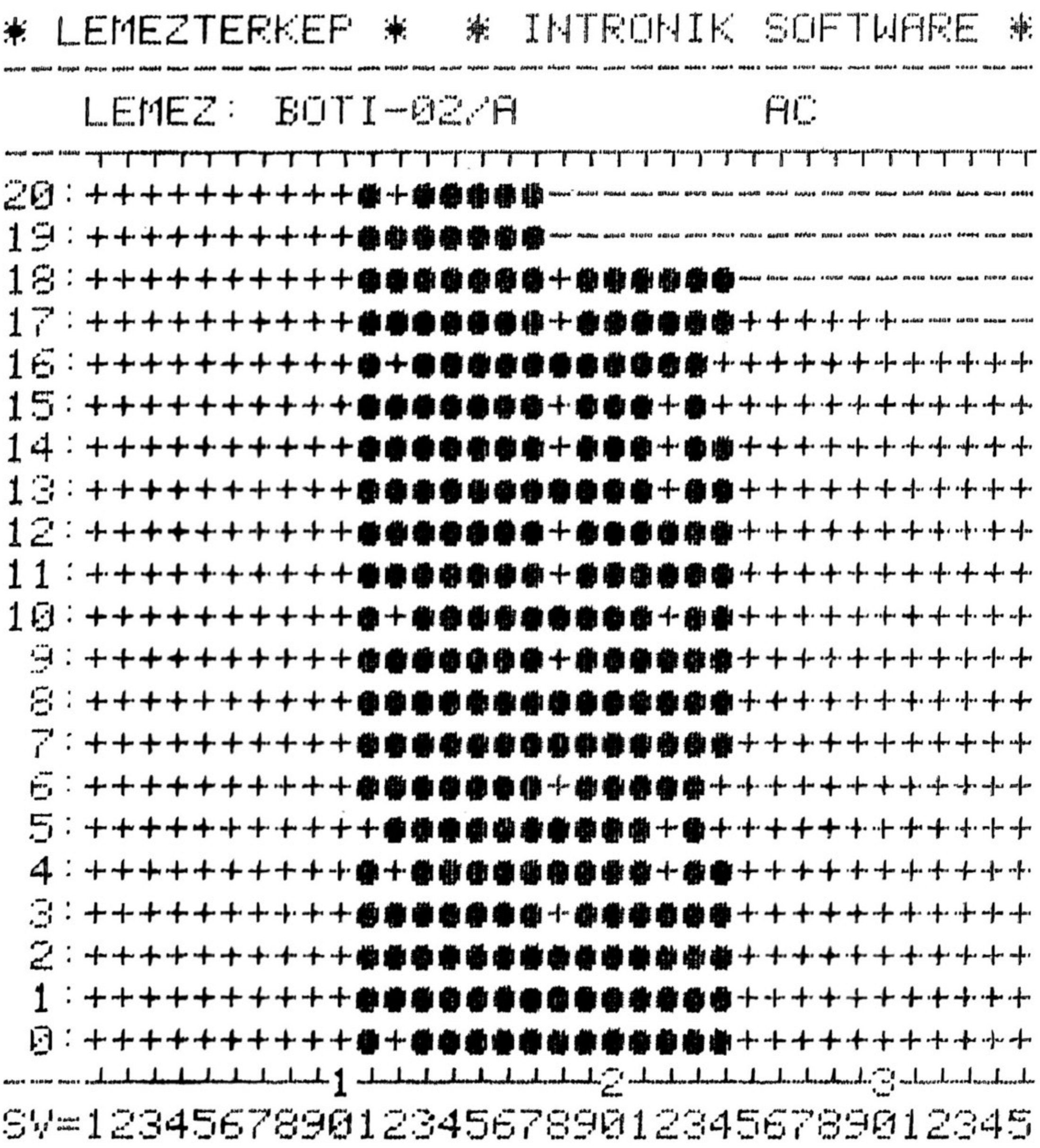
Egészen pontosan:

az 1-től 17-ig terjedő sávokon sávonként 21,
a 18-tól 24-ig terjedő sávokon sávonként 19,
a 25-től 30-ig terjedő sávokon sávonként 18,
a 31-től 35-ig terjedő sávokon sávonként 17

szektor helyezkedik el. *Minden szektor egyforma tárolókapacitású.*
Egy-egy *szektor mérete* 256 bájt, amelyeket konvencionálisan 0-tól 255-ig számozunk. Ebből adattárolásra csak 254 bájt használható fel. A szektor első két bájtjára, a 0-ásra és az 1-esre a lemezkezelő rendszernek van szüksége.
Az egy sávon belüli szektorokat 0-tól 20-ig, 18-ig, 17-ig, illetve 16-ig terjedő számozással, úgynevezett *szektorcímmel* azonosítjuk.

FIGYELEM!
A szektorcímek értelemszerűen csak egy sávon belül azonosítják egyértelműen a szektorokat.

A sávok és a szektorok eloszlását az egyik lemezünkről készített lemeztérképpel illusztráljuk:



3. ábra. Lemeztérkép

Térképünkön a köröcskék jelzik a foglalt, a pluszjelek pedig a szabad szektorokat. (A mínuszjelek a nem létező szektorokat jelölik.) Látható, hogy a lemez feltöltését a lemezkezelő *középről*, pontosabban a 18-as sávtól mindkét irányban kifelé haladva hajtja végre.

Az adatkezelés alapegysége a blokk. Azt az adatmennyiséget nevezzük 1 blokknak, amit a lemezkezelő *egyetlen lemezműveletben* visz fel a lemezre, illetve olvas le onnan. A COMMODORE lemezkezelője 254 bájtos adatblokkokkal dolgozik. Minthogy pontosan ugyanennyi adat fér el a lemez egy adott sávjának egy adott szektorán, a COMMODORE szóhasználatában az egy meghatározott sávcím és szektorcím által együttesen azonosított lemezterületet is blokknak szokás nevezni.

Így egy lemezoldalon összesen 683 blokknyi hely van, amiből formázás után legfeljebb 664 blokk használható adattárolásra. Optimális esetben tehát lemezünkön 168656 bájtnyi adat tárolható. (A gyakorlatban ennél mindig kevesebb adatot vihetünk fel a lemezre. Később majd látni fogjuk, hogy miért.)
Tapasztalatból tudjuk már, hogy a lemezkezelő a lemez formázásakor mindig létrehoz egy tartalomjegyzéket és egy lemeztérképet, amelyeket azután az őket érintő lemezműveletekkel összhangban automatikusan módosít.

Ezek a nyilvántartások mindig a lemez 18-as sávján vannak, amely adattárolásra akkor sem használható, ha a lemeztérkép és a tartalomjegyzék nem tölti ki egészen. (Ezért használhatunk csak 664 blokkot a 683-ból, ugyanis a 18-as sáv pontosan 19 blokkot tartalmaz!) A sávon tárolt információkat, valamint azok tárolási helyét az alábbiakban részletezzük.

A 18-AS SÁV 0-ÁS SZEKTORÁNAK TARTALMA

0–1	bájtokon	:	a következő blokk címe
	tartalma	:	a 0-ás bájton a sávcím = 18
		:	az 1-es bájton a szektorcím = 1
2	bájton	:	formátumkód
	tartalma	:	65 = ASC("A")
3	bájton	:	DOS jelzőkód
	tartalma	:	0
4–143	bájtokon	:	lemeztérkép
	tartalma	:	a lemez minden egyes blokkjához tartozóan, azok címeinek növekvő sorrendjében egy-egy bit, aminek az értéke *0, ha a blokk foglalt,* illetve *1, ha szabad*
144–161	bájtokon	:	lemeznév
	tartalma	:	az a lemeznév, amit formázáskor a NEW parancsban megadtunk; ha rövidebb 16 karakternél, felső állású szóközökkel van feltöltve
162–163	bájtokon	:	lemezazonosító
	tartalma	:	az a lemezazonosító, amit formázáskor a NEW parancsban megadtunk

164	bájton	:	felső állású szóköz
	tartalma	:	160 = ASC(" ")
165	bájton	:	DOS verziószám
	tartalma	:	50 = ASC("2")
166	bájton	:	DOS formátumkód
	tartalma	:	65 = ASC("A")
167–170	bájtokon	:	felső állású szóköz
	tartalma	:	160 = ASC(" ")
171–255	bájtokon	:	nincs kitöltve
	tartalma	:	esetleges

Megjegyezzük, hogy a felső állású szóköz a SHIFT billentyű egyidejű lenyomásával begépelt szóközt jelenti.

A 0-ás szektorból számunkra elsősorban a lemez neve lehet fontos, és esetenként még a lemeztérkép is.

A 18-AS SÁV 1-ES SZEKTORÁNAK TARTALMA

0–1	bájtokon	:	a következő blokk címe
	tartalma	:	a 0-ás bájton a sávcím
		:	az 1-es bájton a szektorcím
2–255	bájtokon	:	tartalomjegyzék
	tartalma	:	legfeljebb 8 darab, egyenként 30 bájtos bejegyzés; mindegyikük egy-egy, a lemezen tárolt állomány vagy program azonosító adataiból áll; részletes leírását lásd külön!

A 18-AS SÁV ÖSSZES TÖBBI SZEKTORÁNAK TARTALMA

0–255	bájtokon	:	következő blokk címe és tartalomjegyzék
	tartalma	:	pontosan ugyanolyan felépítésű, mint az 1-es szektoré

Tehát a 18-as sáv 0-ás szektorában van a *lemeztérkép* és a *lemezfej*, míg a lemez *tartalom jegyéke* az 1-től 18-ig terjedő szektorokban helyezkedik el.
Megjegyezzük, hogy a lemeztérkép számára 140 bájt van fenntartva, ami 1120 bitet jelent. A lemezen 683 fizikai blokk van, tehát a térkép területe jóval nagyobb a szükségesnél.

FIGYELEM!
Minthogy a tartalomjegyzék legfeljebb 18 szektort tölthet ki, és szektoronként legfeljebb 8 bejegyzést tartalmazhat, a lemezen legfeljebb 144 állomány, illetve program tárolható, mert több bejegyzés nem férne el a tartalomjegyzékben.

Most nézzünk meg egy ilyen bejegyzést!

A TARTALOMJEGYZÉK EGY BEJEGYZÉSÉNEK TARTALMA

0	bájton	:	típuskód
	tartalma	:	128, ha a bejegyzés törölt,
		:	129, ha soros állományra,
		:	130, ha programra,
		:	131, ha felhasználói állományra,
		:	132, ha relatív állományra vonatkozik a bejegyzés
1–2	bájtokon	:	az állomány, illetve program kezdőcíme
	tartalma	:	az 1-es bájton a sávcím
		:	a 2-es bájton a szektorcím
3–18	bájtokon	:	állománynév, illetve programnév
	tartalma	:	programoknál az a programnév, amit kimentéskor a SAVE parancsban megadtunk;
		:	állományoknál az az állománynév, amit létrehozáskor az OPEN utasításban megadtunk; ha rövidebb 16 karakternél, felső állású szóközökkel van feltöltve
19–20	bájtokon	:	az első mellékszektor címe (csak REL állománynál)
	tartalma	:	a 19-es bájton a sávcím
		:	a 20-as bájton a szektorcím
21	bájton	:	rekordméret (csak REL állománynál)
	tartalma	:	az a rekordméret, amit a relatív állomány létrehozásakor az OPEN utasításban megadtunk
22–25	bájtokon	:	nincs feltöltve
	tartalma	:	esetleges
26–27	bájtokon	:	a módosított bejegyzés kezdőcíme
	tartalma	:	a 26-os bájton a sávcím
		:	a 27-es bájton a szektorcím csak felülírt programnál, illetve állománynál van kitöltve
28–29	bájtokon	:	az állomány mérete blokkokban
	tartalma	:	a 28-as bájton a blokkok számának alsó bájtja
		:	a 29-es bájton a blokkok számának felső bájtja

Látható, hogy a lemezkezelő számára a lemezen tárolt programok ugyanolyan állományok, mint a többi adatállomány. *A különbség csak a típuskódban van.* Ez az, amitől az egyiket másként kezeli a gép, mint a másikat.

FIGYELEM!
Két ilyen bejegyzés között mindig 2 kihasználatlan „töltelék" bájt van. A 8 bejegyzés így 30+2+30+2+. . . +30 felosztásban pontosan kitölti a szektor 254 bájtját.

Megjegyezzük, hogy ha a lemez tartalomjegyzékét a tárba betöltjük (LOAD), majd kilistázzuk (LIST), akkor a képernyőn nem a fenti típuskódok jelennek meg, hanem az azoknak megfelelő szöveges kódok. Azaz:

törölt bejegyzés esetén	:	DEL
soros állomány esetén	:	SEQ
program esetén	:	PRG
felhasználói állomány esetén	:	USR
relatív állomány esetén	:	REL

Ha már a típuskódoknál tartunk, hadd mutassunk rá egy érdekes jelenségre: a később ismertetendő, úgynevezett *random állományoknak nincs típuskódja.* Ennek az az oka, hogy az ilyen állományok blokkjai csak logikailag képeznek egy egységet, azaz állományt a lemezen. Így a lemezkezelő nem is készít róluk bejegyzést a lemez tartalomjegyzékébe. A lefoglalt blokkokat a lemeztérkép természetesen ettől függetlenül nyilvántartja. (lásd a random állományokkal foglalkozó kötetet!)

FIGYELEM!
A típuskód csak akkor veszi fel a közölt értékeket, ha az állomány megfelelően le volt zárva. Lezáratlan állomány esetén az értéke a megadottnál 128-cal kisebb, azaz 0, 1, 2, 3 vagy 4.

Tudnunk kell még, hogy a lemez tartalomjegyzékébe a bejegyzés már az állomány megnyitásakor bekerül, nyilvánvalóan csak részlegesen feltöltve. Bizonyos információkat ugyanis természetszerűleg csak az állomány lezárásakor lehet a tartalomjegyzékbe bevezetni; például az állomány mérete ilyen. A lemezkezelő ugyancsak a lezáráskor vezeti rá véglegesen a lemeztérképre az állomány által elfoglalt blokkokat.
A „véglegesen" tulajdonképpen azt jelenti, hogy a lemez kezeléséhez a lemezegység a saját (RAM) tárjába átmásolja a lemeztérképet és a tartalomjegyzéket, és a menet közben szükségessé váló módosításokat mindig ezen a másolaton hajtja végre. Az OPEN és a CLOSE hatására pedig ennek a másolatnak a tartalmával módosítja (tulajdonképpen felülírja) a lemezen ténylegesen tárolt nyilvántartást. Így tehát a blokkok lefoglalása folyamatosan megtörténik, de csak a másolaton, és az csak az állomány lezárásakor kerül ki az adathordozóra (a lemezre).
Ha tehát egy állományunk 0 mérettel jelenik meg a tartalomjegyzék listáján, erősen gyanakodhatunk arra, hogy nyitva maradt. Ilyenkor ezt a SEQ szó előtt még egy csillag is jelzi.
A lezárás általában akkor nem következik be, ha valamely lemezművelet elvégzése közben *hiba keletkezett,* vagy még inkább, ha *elfelejtettünk CLOSE parancsot vagy utasítást kiadni.*
A korrekt lezárásra tehát nagyon gondosan ügyeljünk, mert a lezáratlan állományok nem használhatók. Számukra a lemezen nincs lefoglalva hely, akkor sem, ha a bejegyzésük a tartalomjegyzékben (csonkán) szerepel. Ha tehát az állomány fel is került a lemezre, az általa lefoglalt területet a lemezkezelő szabadnak tekinti, és a legközelebbi lemezreírás alkalmával esetleg egészben vagy részben felhasználja.
Megjegyezzük, hogy ugyanez sikertelen kimentés esetén a programokra is igaz. Ilyen

,,lezáratlan'' program-állományok például úgy keletkezhetnek, hogy a SAVE végrehajtása közben a lemezegység ajtaja kinyílik, vagy a lemezen a kimentendő program számára már nincs elég hely stb.

A 18-as sávon tárolt információkra, különösen a lemeztérkép esetén ritkán van közvetlenül szükségünk, hiszen nagy részüket általában nem is befolyásolhatjuk, a gép különben is automatikusan kezeli valamennyit.

A tartalomjegyzékből számunkra érdekes információkat pedig annak kilistázása útján könnyen beszerezhetjük, anélkül, hogy tudnánk, a kérdéses adatok hol és milyen formában vannak tárolva.

Mégis van néhány eset, amikor némelyik információ szemrevételezés útján történő ellenőrzése nem elegendő; célszerűbb azt az emberi beavatkozás kizárásával programra bízni. Ilyenre mutatunk be most három példát. Ezekben használni fogjuk a képernyővezérlés (törlés, inverz írás, kurzormozgatás stb.) programozott formáit. Ilyenkor PRINT utasítást adunk ki, amelyben a vezérlő jelek karakteres konstansok (vagy változók) formájában szerepelnek.

Például ha a nyitó idézőjel után a SHIFT és a CLR gombokat együtt megnyomjuk, majd begépeljük a csukó idézőjelet, egy olyan karakteres konstanst kapunk, amelynek tartalma egy inverz szívecske. Amikor a PRINT utasítás végrehajtódik, nem ez a jel íródik ki, hanem a hatása érvényesül: a képernyő törlődik.

Hasonló módon programozhatók az egyéb vezérlő funkciók is. (Lásd ,,ALAP-ISMERETEK'' 43. és 124. oldal!)

Emlékeztetőül közöljük a fontosabb, és a programjainkban is előforduló vezérlő jeleket, azok begépelési módját és hatását:

```
"■" = [HOME]            : KURZOR BAL FELSO SAROKBA
"■" = [SHIFT]+[CLR]     : KEPERNYO TORLESE
"■" = [CTRL]+[RVS ON]   : ATTERES INVERZ IRASRA
"■" = [CTRL]+[RVS OFF]  : VISSZATERES NORMAL IRASRA
"■" = [CRSR]            : KURZOR LE
"■" = [SHIFT]+[CRSR]    : KURZOR FEL
"■" = [CRSR]            : KURZOR JOBBRA
"■" = [SHIFT]+[CRSR]    : KURZOR BALRA
```

A LEMEZ NEVÉNEK ELLENŐRZÉSE PROGRAMBÓL

Adatfeldolgozási környezetben gyakran lehet szükségünk arra, hogy valamely adatfeldolgozó programunk ellenőrizze, hogy valóban az a lemez van-e behelyezve a lemezegységbe, amely a feldolgozandó állományunkat, betöltendő programunkat tartalmazza.

A lemezkezelő ezt az ellenőrzést nem hajtja végre, így tévedés esetén szerencsések vagyunk, ha például a feldolgozandó állományunkkal azonos nevű nincs a lemezen, vagy ha a létrehozandó állományunkkal azonos nevű már létezik a lemezen. Ilyenkor ugyanis hibajelzést kapunk, és gyorsan észbe kapva kicserélhetjük a lemezt a megfelelőre.
Jobban járunk tehát, ha beleolvasunk a lemez tartalomjegyzékébe, pontosabban a lemezfejbe; ott megkeressük a lemez nevét, majd ellenőrizzük, hogy az azonos-e a programozó által megadott vagy a gépkezelő által begépelt névvel.
Erre az ad lehetőséget, hogy a lemez tartalomjegyzéke programból állományként is használható — természetesen csak olvasásra.
Továbbá tudjuk, hogy a lemez neve a tartalomjegyzék 144-es, tehát a 145-dik bájtján kezdődik, és legfeljebb 16 karakterből állhat úgy, hogy ha rövidebb, felső állású szóközökkel, azaz CHR$(160) karakterekkel pontosan 16 karakteresre van feltöltve.
Végül a COMMODORE BASIC rendelkezik olyan utasítással, amellyel egy ismeretlen állomány bájtonként olvasható. Ez pedig a GET#.

FIGYELEM!
A GET# utasítás nem olvassa be a szektor első két bájtján, a 0-áson és 1-esen tárolt blokkfej információt; vagyis a szektornak csak a programozó számára hozzáférhető 254 adatbájtját tudja olvasni. Ezt a bájtok számlálásakor figyelembe kell vennünk.

Most pedig adjunk NEW parancsot, majd gépeljük be a következő mintaprogramot.
A program bejelentkezéssel indul:

```
104 PRINT "♥"
105 PRINT "  ▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄"
106 PRINT "  ▌ LEMEZAZONOSITAS ▐"
107 PRINT "  ▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀"
```

Bekéri a lemezegység E számát. Ha ez nem 8 vagy 9, újra kéri az adatot.

FIGYELEM!
Ha csak egyetlen lemezegységünk van, mindig a 8-as egységszámot használjuk, mert a program nincs felkészítve az elméletileg lehetséges, de gyakorlatilag hibás adatok kezelésére.

```
110 PRINT:PRINT:PRINT
120 INPUT " ■ EGYSEG? =8/9=";E
130 IF E<>8 AND E<>9 THEN GOTO 120
```

Bekéri a lemez L$ nevét. Ha annak L hossza 1 karakternél rövidebb, azaz nem gépeltünk be semmit, csak a RETURN gombot nyomtuk meg, vagy 16 karakternél hosszabb, akkor újra kéri a nevet:

```
140 PRINT:PRINT:PRINT
150 INPUT "  ☒*■ LEMEZ=";L$
160 L=LEN(L$)
170 IF 1>L OR L>16 THEN GOTO 150
```

Adatállományként megnyitja a lemez tartalomjegyzékét. A megnyitás az OPEN utasítással történik. Először az állományazonosítót adjuk meg. Ezt követi a lemezegység E száma. Majd az adatcsatorna következik, amelyen a lemezegység lebonyolíthatja az adatforgalmat a lemez és a központi egység között.

FIGYELEM!
Semmiképpen ne válasszuk a 15-ös csatornát, mert az a parancs-csatorna, valamint a 0-ást vagy az 1-est, mert azok is kitüntetett csatornák. Vagyis kizárólag adatcsatornát (2–14) választhatunk. Megjegyezzük, hogy ilyenkor is célszerű az állományazonosítót a csatornaszámmal azonosnak választani.

Végül az OPEN negyedik paramétereként meg kell adnunk az állomány nevét. Ez a tartalomjegyzék esetén, mint tudjuk,"$".

FIGYELEM!
Ilyenkor nem kell, sőt nem is szabad a SEQ és READ információkat megadni.

Vigyázzunk, hogy ne keverjük össze az állományazonosítót az állománynévvel! Az előbbi csak a programon belül érvényes, és arra jó, hogy megkülönböztesse a különböző állományokra vonatkozó utasításainkat, ha a program több állományt is kezel. A lemezkezelő azonban mit sem tud a mi ideiglenes állományazonosítónkról, ő az állományt az állománynévvel azonosítja; ez is szerepel a lemezen.

```
210 OPEN 2,E,2,"$"
```

Kipörgeti a lemez tartalomjegyzékének első 142, számunkra most értéktelen adatbájtját, figyelmen kívül hagyva az első két hozzáférhetetlen bájtot. Vagyis az olvasást a 2-es bájttal kezdi:

```
220 V=142: GOSUB 810
```

Beolvassa a tartalomjegyzék fejrészéből a soron következő adatbájttól, vagyis ténylegesen a 144-es bájttól kezdődő 16 bájtot, azaz a lemez formázáskor felvitt N$ nevét:

```
230 V= 16: GOSUB 710
```

Ezek után többet olvasni nem akar, ezért lezárja az állományt, rögtön, ahogy megteheti:

```
240 CLOSE 2
250 PRINT:PRINT:PRINT
```

Összehasonlítja a lemezről leolvasott N$ lemeznevet a billentyűzetről begépelt L$ lemeznévvel:

```
260 IF L$<>N$ THEN GOTO 410
```

Ha a két név azonos, feltűnő OK üzenetet ír ki a képernyőre:

```
310 PRINT " ▣    ■"
320 PRINT " ▣ OK ■"
330 PRINT " ▣    ■"
340 PRINT:PRINT
```

Ezután befejezi a program futását:

```
350 GOTO 599
```

Ha a begépelt L$ és a lemezről leolvasott N$ nevek nem voltak azonosak, a program közli, hogy a gépkezelő által megadott E lemezegységen nem a megadott L$ nevű lemez van:

```
410 E$=STR$(E)
420 E$=RIGHT$(E$,LEN(E$)-1)
430 PRINT " ┌────────────────────────────────┐"
440 PRINT " │ A <";E$;"> EGYSEGEN NEM A";SPC(9);" │"
450 PRINT " │";SPC(30);" │"
460 PRINT " │ <";L$;"> LEMEZ VAN!";SPC(16-L);" │"
470 PRINT " └────────────────────────────────┘"
480 PRINT:PRINT
```

Végül leáll:

```
599 END
```

Az N$ lemeznév beolvasását, illetve az „értéktelen" bájtok kipörgetését *szubrutinokkal, alprogramokkal* oldottuk meg.

A kipörgetés az egyszerűbb; beolvas annyi B$ bájtot, amennyit a V paraméterben megadtunk, de a beolvasott B$ bájtokat nem őrzi meg – azaz mindegyiket felülírja a következővel:

```
810 FOR I=1 TO V
820 GET#2,B$
830 NEXT I
899 RETURN
```

A lemeznév esetén szintén annyi V darab B$ bájtot olvas be, amennyit megadtunk, de ezeket a B$ bájtokat összekapcsolja (konkatenálja) egyetlen karakteres típusú N$ adattá, a lemeznévvé, mégpedig úgy, hogy a 16 karakteresnél rövidebb név végén levő töltelék (felső állású szóköz) karaktereket nem veszi figyelembe:

```
710 N$=""
720 FOR I=1 TO V
730 GET#2,B$
740 IF B$=CHR$(160) THEN GOTO 770
760 N$=N$+B$
770 NEXT I
799 RETURN
```

Ha idáig eljutottunk, a programot azonnal ki is próbálhatjuk! Adjunk RUN parancsot, majd ellenőrizzük egyik lemezünket!

```
LEMEZAZONOSITAS

?? EGYSEG? =8/9=? 8

?? LEMEZ=? TTTTTT
```

```
A <8> EGYSEGEN NEM A
<TTTTTT> LEMEZ VAN!
```

```
LEMEZAZONOSITAS

?? EGYSEG? =8/9=? 8

?? LEMEZ=? BOTI-03/B

OK
```

4. ábra. Lemezellenőrzés

A lemezazonosításnak leginkább akkor van jelentősége, ha több lemezegységünk van, vagy ha bizonyos állományaink több változatban is léteznek. (Vagy ha mindkét eset fennáll.)

A programunk természetesen nem tudhatja, hogy a begépelt L$ és a lemezen levő N$ név közötti különbséget mi okozza. Ugyanazt a hibaüzenetet kapjuk, ha a megadott lemezegységbe nem a megfelelő lemezt tettük be, vagy ha a megfelelő lemezt tettük be ugyan, de nem a megadott lemezegységbe, vagy ha a megadott lemezegységben a megfelelő lemez van, de a program kérdésére a lemeznevet hibásan gépeltük be stb.

Általában természetesen nem ezzel a mintaprogrammal végezzük el a lemez azonosítását, hanem a mintaprogramot átalakítjuk szubrutinná, vagy kiemeljük belőle az azonosításhoz szükséges részeket, és csak azokat építjük be a programunkba; vagy legfeljebb csak a program alapötleteit használjuk fel. Programunk tehát nem arra való, hogy lemásoljuk, hanem arra, hogy tanuljunk belőle — sőt, hogy újabb gondolatokra, ötletekre ösztönözzön.

Végül a jobb áttekinthetőség kedvéért egybefüggően is közöljük a program listáját:

```
100 REM: ================
101 REM: LEMEZAZONOSITAS
102 REM: ================
103 REM:
104 :     PRINT "□"
105 :     PRINT "        _________________"
106 :     PRINT "        ▪ LEMEZAZONOSITAS ■"
107 :     PRINT "        ‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾"
110 :     PRINT:PRINT:PRINT
120 :     INPUT " ▪*■ EGYSEG? =8/9=";E
130 :     IF E<>8 AND E<>9 THEN GOTO 120
140 :     PRINT:PRINT:PRINT
150 :     INPUT " ▪*■ LEMEZ=";L$
160 :     L=LEN(L$)
170 :     IF 1>L OR L>16 THEN GOTO 150
200 REM:
201 REM: ----------
202 REM: AZONOSITAS
203 REM: ----------
210 :     OPEN 2,E,2,"$"
220 :     V=142: GOSUB 810
230 :     V= 16: GOSUB 710
240 :     CLOSE 2
250 :     PRINT:PRINT:PRINT
260 :     IF L$<>N$ THEN GOTO 410
300 REM:
301 REM: ---------
302 REM: HA AZONOS
303 REM: ---------
310 :     PRINT " ▪    ■"
320 :     PRINT " ▪ OK ■"
330 :     PRINT " ▪    ■"
340 :     PRINT:PRINT
350 :     GOTO 599
400 REM:
401 REM: --------------
402 REM: HA NEM AZONOS
403 REM: --------------
410 :     E$=STR$(E)
420 :     E$=RIGHT$(E$,LEN(E$)-1)
430 :     PRINT " ┌──────────────────────────────┐"
440 :     PRINT " │ A <";E$;"> EGYSEGEN NEM A";SPC(9);" │"
450 :     PRINT " │";SPC(30);" │"
460 :     PRINT " │ <";L$;"> LEMEZ VAN!";SPC(16-L);" │"
470 :     PRINT " └──────────────────────────────┘"
480 :     PRINT:PRINT
```

```
500 REM:
501 REM: ----
502 REM: VEGE
503 REM: ----
599 :     END
700 REM:
701 REM: =======
702 REM: OLVASAS
703 REM: =======
710 :     N$=""
720 :     FOR I=1 TO V
730 :        GET#2,B$
740 :        IF B$=CHR$(160) THEN GOTO 770
760 :        N$=N$+B$
770 :     NEXT I
799 :     RETURN
800 REM:
801 REM: ==========
802 REM: KIPORGETES
803 REM: ==========
810 :     FOR I=1 TO V
820 :        GET#2,B$
830 :     NEXT I
899 :     RETURN
```

Megjegyezzük, hogy az alábbi egyszerű módosítással elérhetjük, hogy ha nem a kívánt lemez van az egységben, akkor a program kiírja a meghajtóban levő lemez nevét:

```
400 REM:
401 REM: -------------
402 REM: HA NEM AZONOS
403 REM: -------------
410 E$=STR$(E)
420 E$=RIGHT$(E$,LEN(E$)-1)
430 PRINT " ┌──────────────────────────────┐"
440 PRINT " │ A <";E$;"> EGYSEGEN NEM A";SPC(9);" │"
450 PRINT " │";SPC(30);" │"
460 PRINT " │ <";L$;"> LEMEZ VAN!";SPC(16-L);" │"
470 PRINT " └──────────────────────────────┘"
480 PRINT:PRINT
```

A PROGRAMNEVEK KIGYŰJTÉSE A TARTALOMJEGYZÉKBŐL

Minthogy a GET# utasítással a tartalomjegyzék akár a végéig is kiolvasható, semmilyen elvi nehézséget nem okoz a programnevek kigyűjtése a nyilvántartásból.

Ehhez nem is kell tudnunk, hogy a tartalomjegyzékben pontosan hol is vannak a programnevek. A rájuk vonatkozó bejegyzések a típuskódról ugyanis felismerhetők.

Erről könnyen meggyőződhetünk, ha begépeljük az alábbi mintaprogramot.

A program először is bejelentkezik a képernyőn:

```
110 PRINT "□"
120 PRINT " PROGRAMKERESES"
```

Definiál egy P$ tömböt, a kigyűjtött programnevek számára. Minthogy a tartalomjegyzékben legfeljebb 144 program lehet, ennél nagyobb tömböt felvenni nem érdemes. Kisebbet sem. Ez értelmetlen takarékoskodás lenne, sőt veszélyes is. Ha ugyanis a lemezen netán több program lenne, mint amekkorára a tömböt definiáltuk, az indexhatár átlépésekor a program hibaüzenettel leállna. Márpedig lehetőleg olyan programok írására kell törekednünk, amelyeknél nincs olyan speciális helyzet, amelyben nem működőképesek. Emellett ilyen kis programnál nincs értelme a tárral takarékoskodni, hiszen ilyenkor hely még van bőven, annál is inkább, mert a feltöltetlen karakteres tömbök tömbelemenként csak 3 bájtot foglalnak el.

```
130 DIM P$(144)
```

Nullára állítja be a kigyűjtött programok P számát. Minthogy ezt a RUN parancs hatására a gép automatikusan megteszi, ennek az utasításnak itt legfeljebb dokumentatív szerepe van.

```
140 P=0
```

Megnyit egy állományt, meghatároz egy lemezegységet, és lefoglal egy adatcsatornát a tartalomjegyzék olvasására:

```
210 OPEN 2,8,2,"$"
```

Kipörgeti a 18-as sáv teljes 0-ás szektorát. Mind a 256 bájtot. Ugyanis tudjuk, hogy a tartalomjegyzék bejegyzései az 1-es szektoron kezdődnek.

FIGYELEM!

A GET# még mindig csak az adatbájtokat tudja olvasni, ezért csak 254-szer szabad a kipörgetést végrehajtanunk:

```
220 V=254 : GOSUB 810
```

Beolvassa a soron következő B$ bájtot. Ez az első végrehajtáskor a 18-as sáv 1-es szektorának 2-es bájtja lesz, ami a tartalomjegyzék legelső bejegyzésének a típuskódját tartalmazza:

```
310 GET#2,B$
```

Megvizsgálja, hogy elérte-e a tartalomjegyzék végét. Ez elvileg már a legelső értékes bájt olvasásakor is előfordulhat, hiszen a lemez lehet üres is.

A figyeléshez a már ismert STATUS változót használjuk.
Itt jegyezzük meg, hogy minden blokk első (0-ás) bájtja az állomány (tartalomjegyzék) következő blokkjának sávcímét, a második (1-es) bájtja pedig a szektorcímét tartalmazza, ha van folytatásblokk. Ha nincs, akkor a 0-ás bájt tartalma nulla, az 1-esé pedig a blokk feltöltöttsége — ami azt mutatja, hogy az állományból még hány bájt van hátra. Innen tudja felismerni az állomány végét a lemezkezelő, és ezt használja ki a STATUS beállításánál. Ismételjük, az állományok végén nincs semmilyen külön állományvége (EOF) jel.

```
320 IF STATUS THEN GOTO 610
```

Ha az olvasás során nem értük el az állomány végét, a programunk megnézi, hogy az a B$ bájt, amit beolvasott, egy programra vonatkozó bejegyzés típuskódja-e. Ha ugyanis az, akkor az értékének 130-nak kell lennie.
A vizsgálathoz persze nem magát a beolvasott B$ bájtot, hanem annak az ASCII, azaz az ASC függvénnyel előállított kódját kell használnunk.
Az ASC függvény azonban hibát jelez, ha a paramétere üres karaktersorozat. (Ilyet például úgy tudunk előállítani, hogy egy karakteres konstans nyitó és csukó idézőjele közé nem írunk semmilyen jelet.) A GET# akkor ad ilyen üres karaktert, ha a lemezen levő bájtról hexadecimális nulla értéket olvas be. Minthogy ez előfordulhat, tanácsos a beolvasott B$ bájthoz hozzácsatolni egy "0" karaktert; így ha üres karakter jött be, az ASC függvény "0" paramétert kap, ami nem okoz hibát, ha viszont nem üres karaktert olvastunk be, akkor az ASC egy kéttagú jelsorozatot kap, de ez nem baj, mert a karaktersorozatoknak amúgy is csak az első karakterét veszi figyelembe.

```
330 IF ASC(B$+"0")-128=2 THEN GOTO 510
```

Ha a beolvasott B$ bájton nem egy programra utaló típuskód volt, a programunk újabb B$ bájtot olvas be, és ezt mindaddig teszi, amíg megfelelő (130-as) típuskódot nem talál, vagy el nem éri a tartalomjegyzék végét:

```
410 GOTO 310
```

Ha a beolvasott B$ bájton éppen egy programra utaló típuskód volt, akkor kipörgeti az utána következő két bájtot, amelyek a program kezdőcímét tartalmazzák:

```
510 V=2    : GOSUB 810
```

Ezután beolvassa a soron következő 16 bájtot, vagyis a program nevének a B$ betűit. Ezeket egyetlen N$ programnévvé illeszti össze:

```
520 V=16   : GOSUB 710
```

Amikor megvan a program N$ neve, kigyűjti azt az erre a célra korábban definiált P$ tömbbe, először természetesen 1-gyel megnövelve a kigyűjtött programok P számát, ezt használja a P$ tömb indexelésére is:

```
530 P=P+1
540 P$(P)=N$
```

Ezután újabb programnevek keresésére indul, vagyis tovább olvassa a bájtokat a tartalomjegyzékből, míg csak egy újabb megfelelő típuskódot nem talál, vagy el nem éri az állomány végét:

```
550 GOTO 310
```

Amikor a program a bájtok olvasása után eléri a tartalomjegyzék végét, tájékoztatás végett kiírja a képernyőre a kigyűjtött programneveket. Egyszerű fejléccel, illetve lábléccel jelzi a kiírás elejét, illetve végét.

Minthogy a kigyűjtött programok P számát megőrizte, könnyen meghatározható, hogy a P$ tömbből hány elemet kell kiírnia:

```
610 PRINT "□"
620 PRINT " PROGRAMOK:"
630 PRINT " -----------------"
640 FOR I=1 TO P
650 PRINT " ";P$(I)
660 NEXT I
670 PRINT " -----------------"
```

Befejezésül lezárja az állományt:

```
680 CLOSE 2
```

Végül leáll:

```
699 END
```

A programunk felhasználja az előző programunkban megismert névbeolvasó:

```
710 N$=""
720 FOR I=1 TO V
730 GET#2,B$
740 IF B$=CHR$(160) THEN GOTO 760
750 N$=N$+B$
760 NEXT I
799 RETURN
```

és kipörgető:

```
810 FOR I=1 TO V
820 GET#2,B$
830 NEXT I
899 RETURN
```

szubrutinokat. Az előbbit olyan módosítással, hogy mindig a teljes nevet adja meg, azaz nem hagyja el a felső állású szóközöket.

Mindezek begépelése után tegyünk be egy olyan lemezt, amelynek a tartalomjegyzékét ismerjük, és adjunk RUN parancsot. Ha nem követtünk el gépelési hibát, a programnak működnie kell. Lásd 5. ábra!

A kipróbáláshoz lehetőleg olyan lemezt használjunk, amelyen 10—15 programnál nincsen több, különben a képernyőről ki fog futni a lista, és nem tudjuk ellenőrizni a helyességét.

FIGYELEM!

A mintaprogram csak a lemezen levő programok nevét gyűjti ki, az adatállományokét, ha ilyenek vannak is a lemezünkön, nem veszi figyelembe.

Nézzük meg egyben is a teljes programot!

```
PROGRAMOK:
------------------
LEMEZAZONOSITAS
ZAMATKA
TARTALOMJEGYZEK
PULI
SZOMSZEDOK
ORAREND
SOXOROSIT
DEMOKERESES
DEMOLEMEZAZON
```

5. ábra. Kigyűjtött programnevek.

```
101 :REM >>> BEJELENTKEZES & DEKLARALAS <<<
102 :
110 PRINT "□"
120 PRINT " PROGRAMKERESES"
130 DIM P$(144)
140 P=0
200 :
201 :REM >>> MEGNYITAS & KIPORGETES <<<
202 :
210 OPEN 2,8,2,"$"
220 V=254  :  GOSUB 810 :REM==> KIPORGETES
300 :
301 :REM >>> PROGRAMOK KERESESE <<<
302 :
310 GET#2,B$
320 :   IF STATUS THEN GOTO 610
330 :   IF ASC(B$+"0")-128=2 THEN GOTO 510
410 GOTO 310
510 :   V=2    : GOSUB 810 :REM==> KIPORGETES
520 :   V=16   : GOSUB 710 :REM==> BEOLVASAS
530 :   P=P+1
540 :   P$(P)=N$
550 GOTO 310
600 :
601 :REM >>> KIIRAS & BEFEJEZES <<<
602 :
610 PRINT "□"
620 PRINT " PROGRAMOK:"
630 PRINT " ------------------"
640 FOR I=1 TO P
650 :   PRINT " ";P$(I)
660 NEXT I
670 PRINT " ------------------"
680 CLOSE 2
699 END
```

```
700 :
701 :REM >>> NEV BEOLVASASA <<<
702 :
710 N$=""
720 FOR I=1 TO V
730 :   GET#2,B$
750 :   N$=N$+B$
760 NEXT I
799 RETURN
800 :
801 :REM >>> KIPORGETES <<<
802 :
810 FOR I=1 TO V
820 :   GET#2,B$
830 NEXT I
899 RETURN
```

Természetesen a 330-as sorban a kód megfelelő átállításával elérhetjük, hogy ne a programok, hanem csak az állományok, vagy akár azon belül is csak a soros állományok neve legyen kigyűjtve. Annak sincs akadálya, hogy minden nevet kigyűjtsünk, akár programé, akár állományé is a nyilvántartott név. Sőt, magát a kigyűjtött programnevet is figyelhetjük, így elérhetjük, hogy a program például csak egy meghatározott betűvel kezdődő vagy akár valamilyen más szempontot kielégítő neveket válogassa ki.

A programunk feltételezi, hogy a 18-as sáv 0-ás szektorától fölfelé sehol sem fordulhat elő olyan bájt, a típuskód bájtján kívül, amelyiknek a tartalma megegyezne valamelyik típuskóddal. Ha ilyen eset mégis bekövetkezne, a program kigyűjtene olyan adatokat is, amelyek nem nevek. Ennek a valószínűsége ugyan kicsi, de nem elhanyagolható. Ezért, ha tökéletes biztonságot akarunk, be kell építenünk védelmet. Ez például a bájtok számlálásával és a bájtpozíciók figyelésével megvalósítható. (Ettől eltérő megoldást láthatunk majd a saját tartalomjegyzék készítésénél, néhány oldallal később.)

Adósak vagyunk még egy magyarázattal: mire jó a nevek kigyűjtése. Például azért, mert egy programrendszerünkhöz írhatunk olyan vezérlő programot, amely a lemezről leolvasott programnevekből tudja, hogy milyen programok tartoznak a rendszerhez, és szükség szerint betölti és automatikusan elindítja azt a programot, amelynek a funkciójára a feldolgozás során éppen szükség van.

Vagy gondoljunk például a menüválasztás technikájára! Mennyivel biztonságosabb, ha a választható programneveket nem a gépkezelőnek kell begépelnie, vagy nem konstansként adjuk meg, akár DATA utasításban, akár közvetlen értékadással, hanem azokat a program a menü feltálalásához a lemez tartalomjegyzékéből veszi. Így a menün szereplő programok számának növekedése vagy csökkenése, illetve a programok nevének megváltozása a menüt kiíró vezérlő programban semmilyen változtatást nem igényel. Nem is fordulhat elő, hogy egy lemezen levő program nem szerepel a menüben, vagy megfordítva, kiválasztottunk ugyan egy programot a menüből, de az nem tölthető be, mert már nincs a lemezen, vagy más a neve.

Ehhez a mintaprogramunk önmagában persze kevés, de jó ötleteket ad ahhoz, hogy mi-

lyen programnév- vagy állománynév-kereső eljárásokat építsünk be azokba a programjainkba, amelyeknél ezt szükségesnek látjuk. A mintaprogram mindenesetre bizonyítja, hogy a nevek a tartalomjegyzékből kigyűjthetők.

SAJÁT TARTALOMJEGYZÉK KÉSZÍTÉSE

Noha az előbb bemutatott programunk, akárhány lemezre is próbáltuk ki, soha nem hibázott, a típus figyelésének ténylegesen az az abszolút biztos módja, hogy a bejegyzések szerkezetének ismeretében a típust kizárólag a megfelelő pozíción keressük.

Ha már ismerjük és kihasználjuk a bejegyzések szerkezetét, könnyen írhatunk olyan programot is, ami a programjaink és állományaink minden jellemző adatát megadja — és így többet nyújt a gép által közölt tartalomjegyzéknél.

Itt azt használjuk ki, hogy a tartalomjegyzékben a rendelkezésre álló 18 darab szektoron szektoronként 8 bejegyzés lehet, bejegyzésenként 8 adattal, ahol az adatok a korábban ismertetett táblázat szerinti sorrendben, méretben és tartalommal helyezkednek el.

A program definiálja a C$ címsort, és a sormintaként használt K$ kötőjelet:

```
100 C$=" SAJAT TARTALOMJEGYZEK"
101 K$=" ------------------------"
```

Definiálja a tartalomjegyzék-bejegyzés adatmezőinek M$ megnevezését, valamint V hosszát. (Ilyen adatmezőből bejegyzésenként 8 darab lehet.)

```
110 DIM M$(8),V(8)
120 M$(1)=" TIPUS       : " : V(1)=1
130 M$(2)=" KEZDOCIM    : " : V(2)=2
140 M$(3)=" NEV         : " : V(3)=16
150 M$(4)=" MELLEKSZ    : " : V(4)=2
160 M$(5)=" REKORDMERET : " : V(5)=1
170 M$(6)=" URES        : " : V(6)=4
180 M$(7)=" FELULIRAS   : " : V(7)=2
190 M$(8)=" FAJLMERET   : " : V(8)=2
```

Megnyitja a tartalomjegyzéket mint soros állományt olvasásra, kipörgeti az első 254 bájtot, azaz a 18-as sáv 0-ás szektorát, és ha a tartalomjegyzék üres, leáll:

```
210 OPEN 2,8,2,"$"
220 V=254:GOSUB 810 :REM==> KIPORGETES
230 IF STATUS THEN GOTO 610
```

Ciklust szervez a tartalomjegyzék S szektorainak feldolgozására. (Ilyenből 18 darab van.)

```
310 FOR S=1 TO 18
```

Ezen belül ciklust szervez az egy szektoron belüli T tartalomjegyzék-bejegyzések feldolgozására. (Ezekből szektoronként legfeljebb 8 darab lehet.) Minden bejegyzés feldolgozása előtt törli a képernyőt, és kiírja a fejlécet:

```
320 :  FOR T=1 TO 8
330 :  :  PRINT "♥":PRINT C$:PRINT K$
```

Egy-egy tartalomjegyzék-bejegyzésen belül ciklust szervez az A darab adat beolvasására és kiírására. (Ezekből minden bejegyzésnél mindig pontosan 8 van.)

```
340 :   :   FOR A=1 TO 8
```

Minden egyes adat esetén beolvassa az adatot. Ha a bejegyzés első beolvasott adata N$ nem tartalmaz típuskódot, úgy tekinti, hogy a tartalomjegyzékben nincs több bejegyzés. Ilyenkor a program leáll:

```
350 :   :   :   GOSUB 710 :REM==> BEOLVASAS
360 :   :   :   IF A=1 AND N$=CHR$(0) THEN GOTO 610
```

Ha a bejegyzés létezik, kiírja az adat M$ megnevezését (természetesen mindig a szóban forgó A-dik adatét):

```
370 :   :   :   PRINT:PRINT M$(A);
```

Majd ugyanebbe a sorban kiírja magát az N$ adatot is:

```
400 :   :   :   ON A GOTO 410,420,430,420,410,430,420,480
410 :   :   :   :   PRINT ASC(N$)                                  : GOTO 499
420 :   :   :   :   PRINT ASC(LEFT$(N$,1));",";ASC(RIGHT$(N$,1))   : GOTO 499
430 :   :   :   :   PRINT N$                                       : GOTO 499
480 :   :   :   :   PRINT ASC(LEFT$(N$,1))+ASC(RIGHT$(N$,1))*256   : GOTO 499
```

Megjegyezzük, hogy a nevet és az üres mezőt egy az egyben úgy írja ki, ahogyan beolvasta. A típust és a rekordméretet konvertálnia kell az egybájtos ábrázolásmódról a szokásos numerikus értékre. Ez az ASC függvénnyel valósítható meg. A program vagy állomány kezdőcímét, a mellékszektor címét és a felülíró állomány kezdőcímét szét kell bontani sávcímre (első bájt) és szektorcímre (második bájt). A konvertálás itt sem maradhat el. A program vagy állomány (fájl) méretét pedig át kell számítani az $a+f*256$ képlettel — ahol „a" az alsó bájt, „f" a felső bájt numerikusra konvertált értéke.

Az adat kiírása után áttér a következő A adatra:

```
499 :   :   NEXT A
```

A bejegyzés összes adatának kiírása után megkérdezi, hogy kívánjuk-e kiíratni a soron következő bejegyzés adatait is. Ha a V$ válaszunk igenlő, akkor kipörgeti a bejegyzést követő két kihasználatlan bájtot, ha éppen nem a szektor végén van.

Nemleges V$ válasz esetén feljebb viszi a kurzort, hogy ne fusson ki a képernyőről, majd leállítja a programot.

Bármilyen más válasz esetén új választ kér.

```
510 :   :   PRINT:PRINT K$
520 :   :   PRINT " ?TOVABB? [I/N]"
530 :   :   PRINT K$
540 :   :   V$="":GET V$
550 :   :   IF V$="I" THEN                  GOTO 590
560 :   :   IF V$="N" THEN PRINT ".TTT]":GOTO 610
570 :   :   GOTO 540
```

Ha kértük a kiírás folytatását, áttér a következő T tartalomjegyzék-bejegyzésre:

```
598 :   NEXT T
```

Ha a szektoron belül mind a 8 bejegyzést feldolgozta, áttér a következő S szektorra:

```
599 NEXT S
```

Ha mind a 18 szektor összes bejegyzését feldolgozta, vagy a listázást leállítottuk, lezárja a megnyitott állományt, kiírja a zárósort és leáll:

```
610 CLOSE 2
620 PRINT:PRINT K$
699 END
```

A felesleges B$ bájtok kipörgetése a már ismert szubrutinnal történik, amelyben a kipörgetendő báitok V számát paraméterként kell megadnunk:

```
810 FOR I=1 TO V
820 :  GET#2,B$
830 NEXT I
899 RETURN
```

Az N$ adatok beolvasása is a már ismert módon, B$ bájtonként történik, némi módosítással:

```
710 N$=""
720 FOR I=1 TO V(A)
730 :  GET#2,B$
740 :  IF B$="" THEN B$=CHR$(0)
750 :  N$=N$+B$
760 NEXT I
799 RETURN
```

Először is a rutin a beolvasandó bájtok számát a V(A) tömbből veszi; a megfelelő tömbelemet az adat A sorszáma határozza meg. Másrészt arra ügyelnünk kell, hogy a B$ bájt soha ne tartalmazzon üres karaktert, különben az ASC függvény, ami a konvertáláshoz szükséges, nem lesz végrehajtható. Az üres karakter helyére nem tölthetünk be "0" karaktert, mert ennek a kódja 48. Csak olyan karakter a jó, amelynek az ASCII kódja nulla.

A programot ki is próbálhatjuk. Működésének illusztrálására elegendő néhány részletet bemutatnunk:

```
SAJAT TARTALOMJEGYZEK

TIPUS        :  130
KEZDOCIM     :  17 , 5
NEV          : PULI
MELLEKSZ     :  0 , 0
REKORDMERET  :  0
URES         :
FELULIRAS    :  0 , 0
FAJLMERET    :  28

?TOVABB? [I/N]
```

```
SAJAT TARTALOMJEGYZEK

TIPUS        :  130
KEZDOCIM     :  20 , 13
NEV          : LEMEZAZONOSITAS
MELLEKSZ     :  0 , 0
REKORDMERET  :  0
URES         :
FELULIRAS    :  0 , 0
FAJLMERET    :  6

?TOVABB? [I/N]
```

6. ábra. Példák a saját tartalomjegyzékre

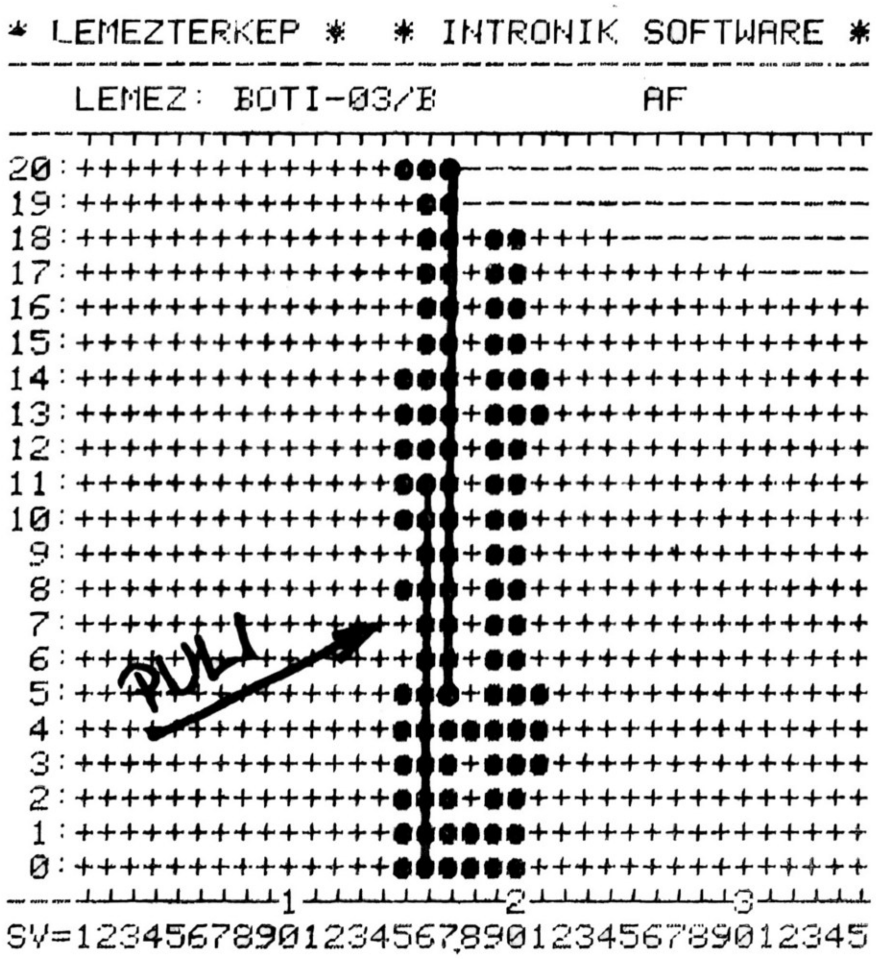
Összehasonlításul közöljük a lemez „hivatalos" tartalomjegyzékét és a lemeztérképet:

```
READY.

0 [illegible]
6     "LEMEZAZONOSITAS"   PRG
7     "ZAMATKA"           PRG
7     "TARTALOMJEGYZEK"   PRG
28    "PULI"              PRG
7     "SZOMSZEDOK"        PRG
9     "ORAREND"           PRG
6     "SOXOROSIT"         PRG
3     "DEMOKERESES"       PRG
6     "DEMOLEMEZAZON"     PRG
3     "EGYPROGRAM"        PRG
3     "KETTESPROG"        PRG
4     "PROGRAMKERESES"    PRG
9     "DEMOTARTALOM"      PRG
566 BLOCKS FREE.

READY.
```

7. ábra. A lemez „hivatalos" tartalomjegyzéke



8. ábra. A vizsgált lemez térképe

A program, szemben az előzőkkel, nemcsak a programok, hanem a soros és a relatív állományok bejegyzéseit is kiírja. Ezt tetszés szerint módosíthatjuk, a típuskód figyelésével, hogy csak az egyiket, vagy másikat vesszük figyelembe. Ízlés szerint „tupírozhatjuk" is, például úgy, hogy csak bizonyos adatokat, mondjuk a nevet, a kezdőcímet és az állományméretet írja ki; vagy hogy a kiírás tetszetősebb vagy tömörebb formában készüljön; hogy a típuskód helyett a típus megjelölését (SEQ, PRG stb.) írja ki; hogy a bejegyzések között meghatározott szempontok szerint válogasson; hogy kívánság szerint a tartalomjegyzéket a nyomtatóra is ki tudja írni — és így tovább.

FIGYELEM!
A program leáll, ha a tartalomjegyzékben nulla típuskódot (lezáratlan törölt bejegyzést) talál. De ha a 360-as sort elhagyjuk, akkor ezeket is kijelzi. Ez esetben viszont csak akkor áll le, ha az összes szektoron végigment, vagy ha a „?TOVABB?" kérdésre nemleges választ adtunk.

Úgy hisszük, mindezeket a módosításokat már az Olvasóra bízhatjuk. Segítségül közöljük a program jelenlegi teljes kódját:

```
READY.
100 C$=" SAJAT TARTALOMJEGYZEK"
101 K$=" ---------------------"
102 REM------------------------------------------------- ADATELOKESZITES
110 DIM M$(8),V(8)
120 M$(1)=" TIPUS        : " : V(1)=1
130 M$(2)=" KEZDOCIM     : " : V(2)=2
140 M$(3)=" NEV          : " : V(3)=16
150 M$(4)=" MELLEKSZ     : " : V(4)=2
160 M$(5)=" REKORDMERET: " : V(5)=1
170 M$(6)=" URES         : " : V(6)=4
180 M$(7)=" FELULIRAS    : " : V(7)=2
190 M$(8)=" FAJLMERET    : " : V(8)=2
200 REM------------------------------------------ MEGNYITAS & KIPORGETES
210 OPEN 2,8,2,"$"
220 V=254:GOSUB 810 :REM==> KIPORGETES
230 IF STATUS THEN GOTO 610
300 REM----------------------------------------------------- FELDOLGOZAS
310 FOR S=1 TO 18
320 :  FOR T=1 TO 8
330 :  :  PRINT "⊐":PRINT C$:PRINT K$
340 :  :  FOR A=1 TO 8
350 :  :  :  GOSUB 710 :REM==> BEOLVASAS
360 :  :  :  IF A=1 AND N$=CHR$(0) THEN GOTO 610
370 :  :  :  PRINT:PRINT M$(A);
400 :  :  :  ON A GOTO 410,420,430,420,410,430,420,480
410 :  :  :  :  PRINT ASC(N$)                                      : GOTO 499
420 :  :  :  :  PRINT ASC(LEFT$(N$,1));",";ASC(RIGHT$(N$,1)) : GOTO 499
430 :  :  :  :  PRINT N$                                           : GOTO 499
480 :  :  :  :  PRINT ASC(LEFT$(N$,1))+ASC(RIGHT$(N$,1))*256 : GOTO 499
499 :  :  NEXT A
510 :  :  PRINT:PRINT K$
520 :  :  PRINT " ?TOVABB? [I/N]"
530 :  :  PRINT K$
540 :  :  V$="":GET V$
550 :  :  IF V$="I" THEN                GOTO 590
560 :  :  IF V$="N" THEN PRINT "⊤⊤⊓":GOTO 610
570 :  :  GOTO 540
590 :  :  IF T<8 THEN V=2:GOSUB 810 :REM==> KIPORGETES
598 :  NEXT T
599 NEXT S
600 REM------------------------------------------------------- BEFEJEZES
610 CLOSE 2
620 PRINT:PRINT K$
699 END
700 REM================================================ ADAT BEOLVASASA
710 N$=""
720 FOR I=1 TO V(A)
730 :  GET#2,B$
740 :  IF B$="" THEN B$=CHR$(0)
750 :  N$=N$+B$
760 NEXT I
799 RETURN
800 REM====================================================== KIPORGETES
810 FOR I=1 TO V
820 :  GET#2,B$
830 NEXT I
899 RETURN

READY.
```

Egyéb lemezkezelési tudnivalók

KÉTLEMEZES FUNKCIÓK

A COMMODORE géphez két lemezmeghajtós (úgynevezett dual) lemezegységek is csatlakoztathatók. Ezekbe egyidejűleg két lemezt is be lehet helyezni.
Ha ilyen lemezegységgel találkozunk, módosítanunk kell a lemezkezelő funkcióinkat. Ilyenkor ugyanis nem elegendő információ a lemezkezelő számára, ha csak a lemezegység logikai egységszámát adjuk meg (esetünkben eddig ez mindig 8 volt), hanem azt is meg kell határoznunk, hogy a lemezegységen belül melyik lemezmeghajtóra vonatkozik a parancs.
A rendszer a lemezmeghajtókat 0, illetve 1 számmal azonosítja. Ezekre kell tehát hivatkoznunk, mégpedig általában a *lemezkezelő funkciót meghatározó kulcsszó után.* Így például a

```
PRINT#15,"NEW0:PROBA,FR"
```

parancs az OPEN utasításban meghatározott számú lemezegység 0-ás meghajtójában levő lemezt formázza, a

```
PRINT#15,"SCRATCH1:FELESLEGES"
```

parancs pedig az 1-es meghajtóban levő lemezről törli a FELESLEGES programot.
Ennél érdekesebb a COPY funkció, ahol a

```
PRINT#15,"COPY0:MASOLAT=1:EREDETI"
```

parancs az 1-es meghajtóban levő lemezről veszi az EREDETI programot, és átmásolja a 0-ás meghajtóban levő lemezre, ahol is MASOLAT néven tárolja.
Ugyanez a lehetőség kihasználható a soros adatállományok összemásolásánál is.
Egylemezes (single drive) lemezegységek esetén, mint amilyen az 1541 is, ezeknek a meghajtó-azonosítóknak a használata szükségtelen, de nem tilos. Ha például programáthelyezhetőségi okokból ragaszkodnunk kell használatukhoz, akkor ezt megtehetjük, de mindenütt kizárólag csak a 0-ás meghajtóra hivatkozhatunk.

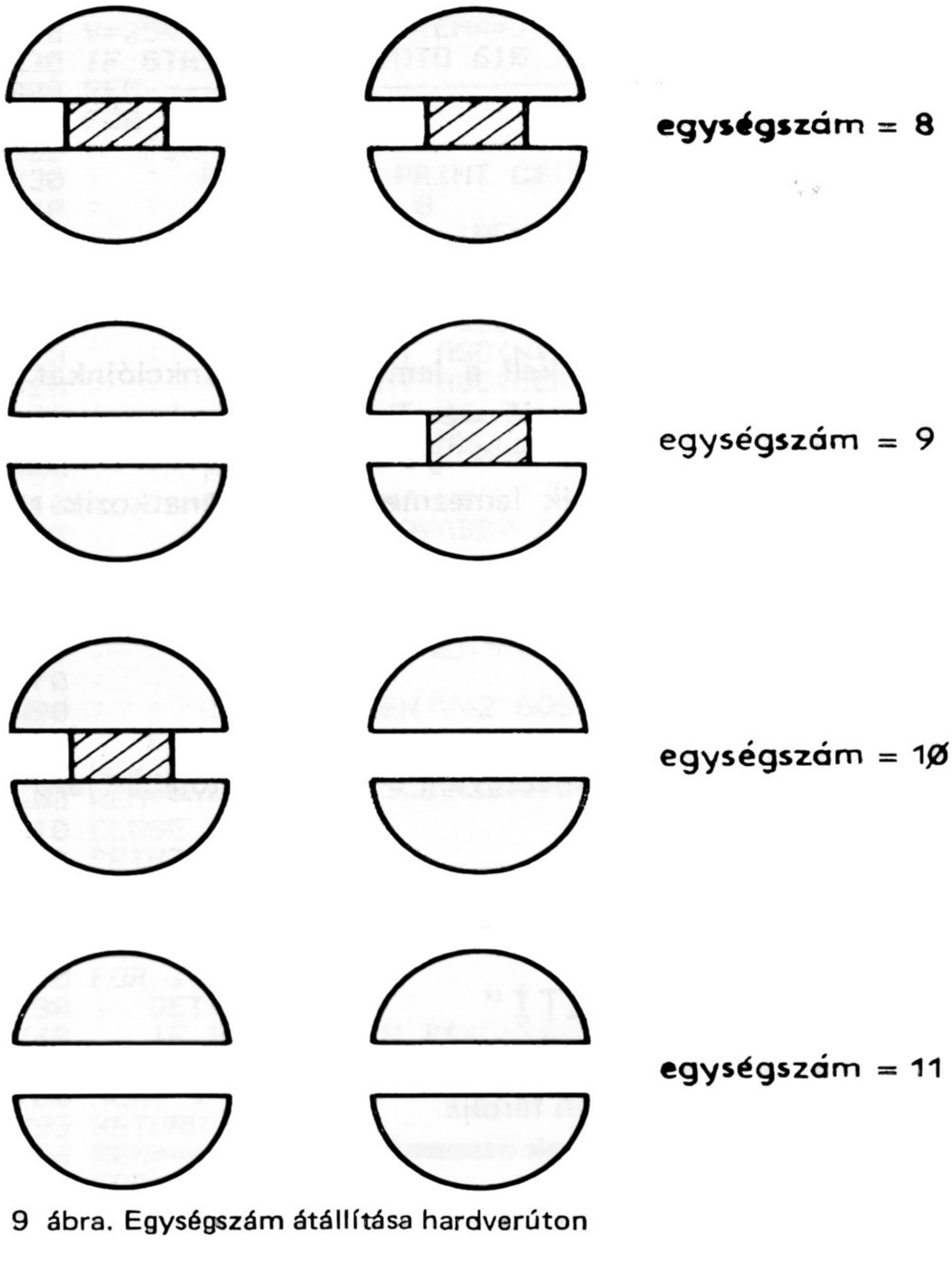
TÖBB LEMEZEGYSÉG KEZELÉSE

A COMMODORE legfeljebb 4 lemezegységet tud kezelni, ha ezek között nincs azonos egységszámú.

Minthogy a lemezegységeket gyárilag 8-as egységszámra beállítva hozzák forgalomba, az átállításról a felhasználónak kell gondoskodnia:

– Először minden kábelt ki kell húzni a lemezegységből.
– Majd le kell szerelni a műanyag dobozfedelet, és azon belül a fémburkolatot is.
– Ezek után a kívánt egységszámnak megfelelő kombinációban át kell vágni az egységszámot meghatározó érintkezőket.
– Végül vissza kell szerelni a fém- és a műanyag burkolatot.

Az átvágások az alábbi kombinációkban hajthatók végre:



9 ábra. Egységszám átállítása hardverúton

Az átvágással az egységszámot egyszer és mindenkorra átállítottuk.

FIGYELEM!
A hardverúton történő átállításhoz egy csillagfejű csavarhúzó, egy kés, valamint szakismeret szükséges. Ezt célszerűen bízzuk szakemberre.

A hardverben beállított egységszám szoftverrel felülbírálható.

Szükség esetén, ha például a 8-as lemezegységünk elromlik, és a javítás ideje alatt egy másik, mondjuk 9-es egységszámú lemezegységgel kívánjuk ideiglenesen helyettesíteni, az egységszám szoftverrel történő átállításával elérhetjük, hogy azokat a programjainkat is használhatjuk, amelyek fixen a 8-as egységszámra vannak beállítva, és azon változtatni nem tudunk, vagy nem akarunk.

Az átállítás úgy kezdődik, hogy bekapcsolás után megnyitjuk a parancs-csatornát:

```
OPEN 15,9,15
```

Majd lemezparancsot adunk ki a lemezegység tárjának felülírására:

```
PRINT#15,"M-W:";CHR$(119)+CHR$(0)+CHR$(2)
+CHR$(40)+CHR$(72)
```

Végül lezárjuk a parancs-csatornát:

```
CLOSE 15
```

Ezek a parancsok programból, utasításként is kiadhatók, és más lemezegységre is végrehajthatók: a PRINT# lemezvezérlő adatsorozatának az utolsó két karaktere mindig a kívánt egységszámnál 32-vel, illetve 64-gyel nagyobb; tehát 9 esetén 41 és 73, 10 esetén 42 és 74, 11 esetén 43 és 75.

FIGYELEM!

Ez az átállítás a lemezegység kikapcsolásáig tart, és akkor visszaáll az eredeti, hardverben beállított egységszám.

```
LOAD "$",8

SEARCHING FOR $
LOADING
READY.
OPEN 15,8,15

READY.
PRINT#15,"M-W:";CHR$(119)+CHR$(0)+CHR$(2)+CHR$(41)+CHR$(73)

READY.
CLOSE 15

READY.
LOAD "$",8

SEARCHING FOR $
?DEVICE NOT PRESENT  ERROR
READY.
LOAD "$",9

SEARCHING FOR $
LOADING
READY.
```

A LEMEZMŰVELETEK SIKERESSÉGÉNEK LEKÉRDEZÉSE

Az adatállományokkal kapcsolatban felhívjuk a figyelmet a parancs-csatorna már ismert lekérdezési lehetőségének kibővített változatára. Ha bármilyen lemezművelet után kiadunk egy

```
8888 INPUT#15,H,H$,SA,SZ
```

utasítást, akkor a parancs-csatornáról leolvashatjuk a lemezművelet elvégzésének sikerességére vagy sikertelenségére vonatkozó információkat, amelyek a megadott változókban jelennek meg:

H : hibakód
tartalma = 0, ha a lemezművelet sikeres volt;
= 20–74, sikertelen lemezművelet után;
= 1–19 esetén nem jelent hibát;

H$: hibaüzenet
tartalma = a hibakódnak megfelelő (angol nyelvű) hibaüzenet, például 0 esetén „OK";

SA: sávcím

SZ: szektorcím

```
10 OPEN 15,8,15
20 OPEN 2,8,2,"PROBA,SEQ,READ"
30 INPUT#15,H,H$,SA,SZ
40 PRINT:PRINT" HIBAKOD =" ;H
50 PRINT:PRINT" HIBA     = ";H$
60 PRINT:PRINT" SAV      =" ;SA
70 PRINT:PRINT" SZEKTOR =" ;SZ
80 CLOSE 2
90 CLOSE 15


READY.
```

```
RUN

 HIBAKOD = 62
 HIBA    = FILE NOT FOUND
 SAV     = 0
 SZEKTOR = 0
```

A *sávcím* és a *szektorcím együttesen* egy *blokkcímet* határoz meg. A lemezművelet jellegétől, esetenként azon belül a hiba típusától függ, hogy itt milyen blokkcímet kapunk. Sikeres lemezművelet esetén mindig SA=0 és SZ=0 jelenik meg.

Leggyakrabban annak a blokknak a címét adja meg a lemezkezelő, amelynek a kezelése, írása/olvasása közben a hibát felfedezte, tehát nem azét, amelyikben a hiba ténylegesen előfordult. Az esetek döntő többségében ez a kettő megegyezik. Számítsunk azonban arra is, hogy néha az utolsó sikerrel kezelt blokk címét kapjuk meg.
Ha a lemezkezelő nem tudja megállapítani a hibás blokk címét, vagy még inkább, ha a lemezkezelés során még egyáltalán nem nyúlt blokkhoz, akkor SA=0 és SZ= 0 értéket ad meg.
Előre már most jelezzük, hogy ennek a visszakapott blokkcímnek az úgynevezett random állományok kezelésekor sajátos és rendkívül fontos szerepe lesz. (A random állományokat külön kötetben fogjuk tárgyalni.)

FIGYELEM!
A parancs-csatorna csak programból kérdezhető le. Az INPUT# ugyanis parancsként nem adható ki. (Lásd „ALAPISMERETEK", 81. oldal!)

MEGJEGYZÉSEK A LEMEZKEZELÉSHEZ

A NEW, a VALIDATE, az INITIALIZE lemezkezelő parancsok mindig a lemezre vonatkoznak, de az adatállományokra is kihatásuk van. A VALIDATE törli a lemezen esetleg fent levő random szervezésű adatállományt, a NEW pedig formázáskor végigírja és vissza is olvassa a teljes lemezt, így az összes állományt megsemmisíti. Az INITIALIZE hatására pedig az éppen nyitva levő adatállományok megsemmisülhetnek.
A LOAD, SAVE, VERIFY parancsokkal csak programot lehet betölteni, kimenteni, ellenőrizni. Adatállományokra nem használhatók.
Ez alól kivétel a LOAD, amellyel a lemez tartalomjegyzéke is betölthető.
A RENAME, SCRATCH, COPY parancsokkal mind programokat, mind soros, illetve relatív szervezésű adatállományokat lehet átnevezni, törölni, illetve másolni. Ez alól kivétel a COPY funkció összemásolni tudó változata, amely csak soros adatállományokra alkalmazható.
Megjegyezzük, hogy a COPY működik kétlemezes (dual) egységen is, tud másolni az egyik meghajtóból a másikba, de nem másolhatunk vele egyik lemezegységről másik egységre.
Sem a COPY, sem a RENAME nem hajtható végre megnyitott állapotban levő állományokra.

A SOROS ÁLLOMÁNYOKRA VONATKOZÓ HIBAÜZENETEK

OK (hibakódja = 0)

Minden rendben.

READ ERROR (hibakódja = 20,21,22,23,24,27)

Olvasási hiba. Leggyakrabban a lemeztartalom megsérülése okozza. Ilyenkor próbáljuk meg újraformázni a lemezt. Ha egy másik lemezegységen nem sikerül, a lemezt eldobhatjuk. Ugyanez a teendő, ha maga a lemez sérült meg. A hibát okozhatja a lemezegység meghibásodása is. Gyakori hibaok lehet még: a lemez-

egységben nincs lemez; a lemezegységben levő lemez nincs formázva; a lemez nincs jól behelyezve; a lemezegység ajtaja nincs jól becsukva, esetleg menet közben kinyílt.

WRITE PROTECT ON (hibakódja = 26)

Írásvédelemmel ellátott lemezre adtunk ki lemezre író utasítást. Vagy cseréljük ki a lemezt, vagy távolítsuk el róla az írásvédő címkét.

WRITE ERROR (hibakódja = 25,28)

Íráshiba. Általában lemezsérülés vagy a lemezegység meghibásodása okozza. Próbálkozzunk másik lemezzel vagy lemezegységgel.

SYNTAX ERROR (hibakódja = 30,31,32,33,34,39)

Formailag hibás, értelmezhetetlen lemezparancsot adtuk ki. Nézzük át a lemezparancsokat, keressük meg, és javítsuk ki a hibát.

RECORD NOT PRESENT (hibakódja = 50)

Nincs adatrekord. Csak *olvasáskor* kaphatjuk ezt az üzenetet, amikor az olvasás túlhaladt az állomány végén.

WRITE FILE OPEN (hibakódja = 60)

Megnyitási hiba. Olyankor kapjuk, ha egy írásra megnyitott állományt nyitva felejtettünk, majd olvasásra akarjuk megnyitni. Ha sikerül az állományt lezárnunk, a hiba megszűnik.

FILE NOT OPEN (hibakódja = 61)

Az állomány, amelyből olvasni, vagy amelybe írni akartunk, nincs megnyitva. Ilyen hiba esetén a lemezkezelő figyelmen kívül hagyja az író/olvasó utasítást. A hiba az állomány megfelelő megnyitásával megszüntethető. Ritkábban az okozza, hogy elfelejtettünk OPEN utasítást kiadni, gyakrabban inkább az, hogy az OPEN-ben és az I/O utasításban megadott állományazonosító nem egyezik meg.

FILE NOT FOUND (hibakódja = 62)

Az állomány nincs a lemezen. Leggyakrabban az okozza, hogy az állomány nevét tévesen adtuk meg, hibásan gépeltük be. Ugyancsak gyakori ok, hogy nem a megfelelő lemezt tettük a lemezegységbe.

FILE EXISTS (hibakódja = 63)

Az állomány már létezik. Olyankor kapjuk, ha írásra nyitunk meg egy állományt, de annak a neve a lemez tartalomjegyzékében *már szerepel.* Vagy az állomány nevét nem választottuk meg jól, vagy nem a megfelelő lemezt tettük be.

FILE TYPE MISMATCH (hibakódja = 64)

Az állomány típusa nem megfelelő. Ha a megnyitandó állomány neve szerepel a lemez tartalomjegyzékében, de ott az állomány típusa nem soros (SEQ), akkor kapunk ilyen üzenetet. Hasonlóan az előbbiekhez, vagy a nevet, vagy a lemezt választottuk meg rosszul.

NO CHANNEL (hibakódja = 70)

Nincs szabad csatorna. Megnyitáskor kapjuk ezt az üzenetet, ha az állományhoz kijelölt csatornát már egy másik (megnyitott) állomány használja, vagy ha már öt adatállomány van nyitva. Az utóbbi esetben a hatodik soros állományt már nem lehet megnyitni, mivel ezek az adatállományok legfeljebb 5 csatornát használhatnak. Leggyakrabban az okozza ezt a hibát, hogy egy vagy több állományt nem zártunk le.

DIRECTORY ERROR (hibakódja = 71)

Megsérült, felülíródott a lemeznyilvántartás — a tartalomjegyzék vagy a lemeztérkép, vagy mindkettő. Adjunk ki INITIALIZE parancsot, ettől a hiba általában elmúlik. A nyitva maradt állományok ilyenkor megsemmisülhetnek. Ha az INITIALIZE sem hatásos, a lemezegységet kapcsoljuk ki, majd kapcsoljuk újra be.

DISK FULL (hibakódja = 72)

A lemez megtelt. Nemcsak akkor kapjuk, ha a lemezen már nincs több szabad blokk, hanem akkor is, ha (például sok apró állomány esetén) a lemez tartalomjegyzékébe már több bejegyzést felvinni nem lehet. Ha nem az utóbbiról van szó, esetenként a lemez tömörítésével (VALIDATE) elegendő helyet szabadíthatunk fel a hiba elhárításához.

DRIVE NOT READY (hibakódja = 74)

A lemezegység nem üzemkész. Oka: a lemezegységben nincs lemez. A parancs-csatornáról akkor is ezt az üzenetet kapjuk, ha a LOAD esetén a központi egység FILE NOT FOUND üzenettel reagál. Ugyanezt az üzenetet kapjuk, ha a lemez nem jól van behelyezve az egységbe, vagy az ajtó nyitva van. *FIGYELEM!* Ha rossz egységszámot adtunk meg, vagy a lemezegység nincs bekapcsolva, vagy nincs a központi egységgel összekötve, nem ezt, hanem a DEVICE NOT PRESENT (nemlétező eszköz) üzenetet kapjuk, de nem a parancs-csatornán, hanem programmegszakítás után közvetlenül a központi egységtől.

A COMMODORE ADATÁLLOMÁNYAI

A lemezkezelő háromféle adatállomány létrehozására, módosítására, karbantartására, törlésére, egyszóval kezelésére képes. Ezek:

- a soros adatállományok,
- a random adatállományok,
- a relatív adatállományok.

Mindenekelőtt meg kell különböztetnünk az állományok tárolási, szervezési és elérési módját. A tárolási módot az határozza meg, hogy az állomány elemei, a rekordok az adathordozón, azaz a lemezen vagy a mágnesszalagon fizikailag hogyan vannak egymáshoz képest elhelyezve:

- sorjában egymás után, azaz sorosan, mint a földalatti kocsijai az alagútban, vagy
- össze-vissza, látszólag véletlenszerűen, mint a nézők egy erősen foghíjas nézőtéren,

Az elérési mód attól függ, hogy miként tudunk az állomány elemeihez, a rekordokhoz hozzáférni:

- sorban egymás után, azaz sorosan, mint a gyöngyökhöz egy füzéren, vagy
- közvetlenül, mint egy tálra kitett sütemények esetén.

A szervezési módot az határozza meg, hogy milyen tárolási mód milyen elérési módot tesz lehetővé, vagyis az, hogy milyen kapcsolat van az állomány egyes rekordjai, valamint azok tárolási helye között.

A *soros szervezésű állomány* rekordjai sorban egymás után, érkezési sorrendben vannak tárolva, és csak ebben a sorrendben, sorban egymás után érhetők el. Ilyen állomány akár lemezen, akár mágneskazettán, akár nyomtatón létrehozható.

A *random szervezésű állomány* rekordjai véletlenszerűen vannak tárolva, vagyis a lemezen

bárhol előfordulhatnak, és közvetlen eléréssel kezelhetők. Ez az állomány értelemszerűen csak lemezen valósítható meg.

A *relatív szervezésű állomány* rekordjai meghatározott sorrendben egymás után vannak tárolva, de nemcsak sorosan, hanem mindegyikük külön-külön közvetlenül is elérhető. Ilyen állományok is csak lemezen létezhetnek.

A *közvetlen elérést biztosító* (random és relatív) állományoknak külön kötetet szenteltünk a sorozatunkban. Könyvünk második részében csak a soros állományokkal foglalkozunk.

II. RÉSZ
SOROS ÁLLOMÁNYOK

Alapműveletek soros állományokkal

A soros adatállományok már a számítástechnika legkorábbi fejlődési szakaszában megjelentek. Szerepük és jelentőségük akkor volt a legnagyobb, amikor a külső adattárolás legfőbb eszközei a mágnesszalagos egységek voltak.
A közvetlen elérést lehetővé tevő (úgynevezett DASD = Direct Access Storage Device) háttértárak, elsősorban a lemezek elterjedésével a soros állományok szerepe csökkent.
Ez a megállapítás kifejezetten érvényes a COMMODORE gépre, ahol a kazettás egység, noha elvileg alkalmas soros adattárolásra, gyakorlatilag üzemszerű adatfeldolgozásra nemigen használható, ugyanakkor a lemezegység „kulturált'' állománykezelési lehetőséggel rendelkezik. Könyvünkben éppen ezért csak a *lemezen létrehozott soros* adatállományokkal foglalkozunk.
Manapság, az adatbázisok, információs rendszerek korában a soros állományok egy immár letűnt kor emlékeinek tűnnek, és óhatatlanul felmerül a kérdés, hogy érdemes-e egyáltalán velük foglalkoznunk, főképpen a COMMODORE esetében, ahol olyan egyéb állományok is léteznek, amelyekre a közvetlen elérés előnyei mellett a soros feldolgozás is megvalósítható.
Itt meg kell jegyeznünk, hogy a COMMODORE—64 gépet *nem adatfeldolgozásra tervezték.* Ez meghatározza a hardver és szoftver lehetőségeit. A hajlékony lemez tárolókapacitása kicsi, az adatátvitel lassú, a működése közel sem olyan üzembiztos, mint a kimondottan adatfeldolgozási célokra készült lemezegységeké; az állományok, különösen a közvetlenül elérhető állományok kezelése ugyan nem túl nehéz, de meglehetősen körülményes. Mindezek miatt a COMMODORE—64 gépre komoly adatfeldolgozó rendszereket megvalósítani meglehetős kockázattal jár — ha egyáltalán lehetséges.
Ennek ellenére az adatfeldolgozás alapkérdéseivel mindenképpen foglalkoznunk kell, mégpedig két okból.
Egyrészt a soros feldolgozás olyan alapvető programozási elemekre, fogásokra, algoritmusokra épül, amelyek más feldolgozási módot alkalmazó programoknak is szerves építőkövei lehetnek — legalábbis a hagyományos magas szintű programnyelveken történő megvalósítás esetén. Így ezek az elvek még ma is a számítástechnika alapismereteihez, mintegy szakmai minimumához tartoznak.
Másrészt vannak olyan esetek, amikor a soros adattárolás és a soros feldolgozás természe-

tes, esetleg az egyedüli lehetséges, de legalábbis a kézenfekvő és legolcsóbb megoldás. (Gyakran ez a helyzet azokkal az adatokkal, amelyek időrendi sorrendben folyamatosan egymás után keletkeznek, és kerülnek be a feldolgozásba. Erre példa lehet egy olyan rendszer, amelyben eladott árukról kell szállítólevél alapján számlát készítenünk.)

A soros állományok legnagyobb *hátránya,* hogy az egyébként könnyű kezelhetőségük ellenére a karbantartásuk igen körülményes, egy bizonyos rekordot csak több rekord olvasása árán lehet elérni, a soros állományok rendezése pedig lassú és tárigényes művelet. A probléma kisebb, ha elegendően nagy és megfelelően gyors háttértárral, valamint jó rendezőprogrammal, vagy legalábbis jó rendezési eljárással rendelkezünk.

A karbantartás bonyolultsága azonban a soros adattárolás jellegéből fakad; a gépi lehetőségek és korlátok alig változtatnak rajta. A karbantartó eljárás tehát nem a COMMODORE tulajdonságai miatt bonyolult, ugyanilyen lenne bármely más, akár nagy gépen is.

Ha tehát a soros feldolgozás menetéből a karbantartás és a rendezés kiküszöbölhető, vagy csak ritkán van rájuk szükség, illetve nem túl sok adatra hajtandók végre, akkor a soros felvitel, adattárolás és -feldolgozás számos gépesítendő feladat egyszerű és olcsó megoldása lehet.

Éppen a mikrogépek sajátosságai miatt, illetve azok kihasználásával és ügyes szervezéssel a soros feldolgozás kényelmesen megvalósítható. (Gondoljuk át a már említett számlázási példát; ott a rendezésre nincs feltétlenül szükség, a karbantartás pedig az interaktív felvitelbe beépített, mindenre kiterjedő adatvizsgálattal kiváltható, vagy legalábbis lényegesen leegyszerűsíthető.)

Minderre könyvünk hátralevő részében elegendő példát mutatunk be. Ezeket azzal a meggyőződéssel adjuk közre, hogy általuk az olvasó nemcsak számítástechnikai, programozási ismereteit bővítheti, hanem a *mindennapi gyakorlatba átültethető tapasztalatokkal is gazdagabb lesz.*

Nem tagadjuk, hogy noha a könyvünkben bemutatott mintaprogramokat (néhány kivétellel) létező és ma is működő mikrogépes (tehát nem nagygépes!) adatfeldolgozó rendszerek programjaiból válogattuk ki és egyszerűsítettük le, azok egyike sem COMMODORE gépen üzemel.

Joggal kérdezhetjük, miért érdemes a COMMODORE adatfeldolgozási lehetőségeivel részletesen foglalkoznunk? A válasz egyszerű.

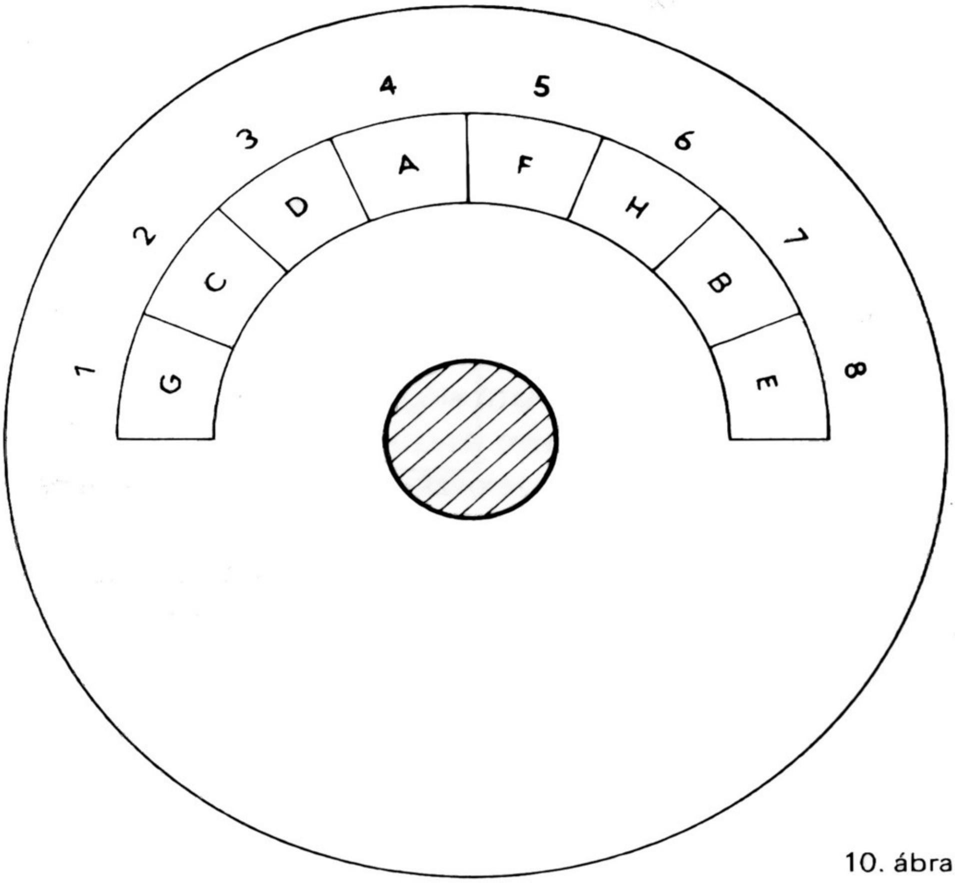
Ámbár a COMMODORE nem adatfeldolgozó gép, rendkívül kiváló oktatóeszköz. Alkalmas arra, hogy az adatfeldolgozás alapelveit, fogásait, algoritmusait rajta megtanuljuk, és 10–30 rekordos próbaállományokkal kipróbáljuk és begyakoroljuk. Így az adatfeldolgozás lényegére vonatkozó, olcsón és könnyen megszerzett tapasztalatainkat más gépen és környezetben is kamatoztathatjuk.

Másrészt a COMMODORE, kis szervezeti egységekben, meghatározott korlátok között, alkalmas lehet helyi jellegű és célú feldolgozások végrehajtására — aminek csak annyi a célja, hogy előkészíti a talajt egy szélesebb körű számítógépes feldolgozás bevezetésére.

SOROS ÁLLOMÁNY

A soros állomány a lemezen egy olyan lemezterületet foglal el, amelynek fizikai mérete legfeljebb 168656 bájt lehet. Egy lemezoldalon legfeljebb 144 soros állomány tárolható.

Az *adatok* az állományba általában nem egyenként, hanem valamilyen szempont szerint úgynevezett *rekordokba csoportosítva kerülnek* fel. (Elvileg nincs akadálya annak, hogy egy rekord csupán egyetlen adatot tartalmazzon.)



10. ábra. Soros állomány

A rekordok előfordulásuk sorrendjében követik egymást a lemezen, és csak ugyanebben a sorrendben dolgozhatók fel az állomány egyetlen végigolvasása során.

Fontos tudnivaló, hogy a rekordba szervezett adatok között elhatárolójeleknek kell lenniük, ha nem akarjuk, hogy a rekord adatai egyetlen folyamatos jelsorozattá olvadjanak össze.

A COMMODORE egyidejűleg *legfeljebb 5 soros adatállományt* tud kezelni, függetlenül a lemezegységek számától.

Egy program természetesen ennél több soros állományt is kezelhet, de azok közül egyszerre legfeljebb ötnek az adatcsatornája lehet nyitva.

A soros állományok kezelésére egy konkrét példán keresztül mutatunk be szemelvényeket.

Tegyük fel, hogy nyilvántartást akarunk létesíteni a költségviselőinkről, melyekből nincsen túl sok, mondjuk legfeljebb 99 lehet.

Egy költségviselőről az alábbi adatokat kívánjuk tárolni:

- kódja (pontosan 4 betű)
- megnevezése (legfeljebb 30 karakter)
- tervezett ráfordítás (legfeljebb 9999)
- tényleges ráfordítás (legfeljebb 9999)

A számadatok értelemszerűen nem lehetnek negatívak, és tetszésünk szerinti egységben (forint, munkaóra, munkaegység, emberhónap stb.) értendők.

Célszerű, hogy az egy-egy költségviselőre vonatkozó adatok képezzenek az állományban egy-egy rekordot:

- K$ (kód)
- sorvége jel

– M$ (megnevezés)
– sorvége jel
– T (tervezett ráfordítás)
– sorvége jel
– R (tényleges ráfordítás)
– sorvége jel

Már említettük, hogy az adatok összeolvadását elhatárolójelekkel (szeparátorjelekkel) akadályozzuk meg. Válasszuk erre a célra – az I. részben említett okokból – a sorvége (kocsi vissza) jelet, amely vagy a CHR$(13) függvényértékkel, vagy (értelemszerűen csak a rekord végén) a RETURN gomb megnyomásával állítható elő.

A lemezen ilyenkor a rekord, mint egység, fizikailag nem létezik; csak számunkra csoportosítja a logikailag összetartozó adatokat, amelyekkel a lemezkezelő a blokkot folyamatosan tölti fel.

Így akár töredék rekord is, például egy fél rekord is írható, illetve olvasható. Sőt, a felírás/leolvasás történhet akár adatonként is.

LÉTREHOZÁS

A létrehozás lényege, hogy beszerezzük, például más állományokból kiolvassuk, vagy billentyűzetről begépeljük az állományhoz tartozó adatokat; azokat rekordba csoportosítjuk, majd a rekordokat felvisszük az állományba (11. ábra).

Ügyelnünk kell azonban arra, hogy a lemezre lehetőleg ne kerüljenek hibás adatok, mert (amint látni is fogjuk) azok kijavítása igen költséges és időigényes. Ezért, különösen a billentyűzetről bevitt adatok ellenőrzése rendkívül fontos.

Most nézzük magát a programot. Adjunk ki NEW parancsot, és gépeljük be az alábbi programrészleteket, amelyek együttesen egy kipróbálható, futtatható komplett programot képeznek.

Mindenekelőtt a program bejelentkezik:

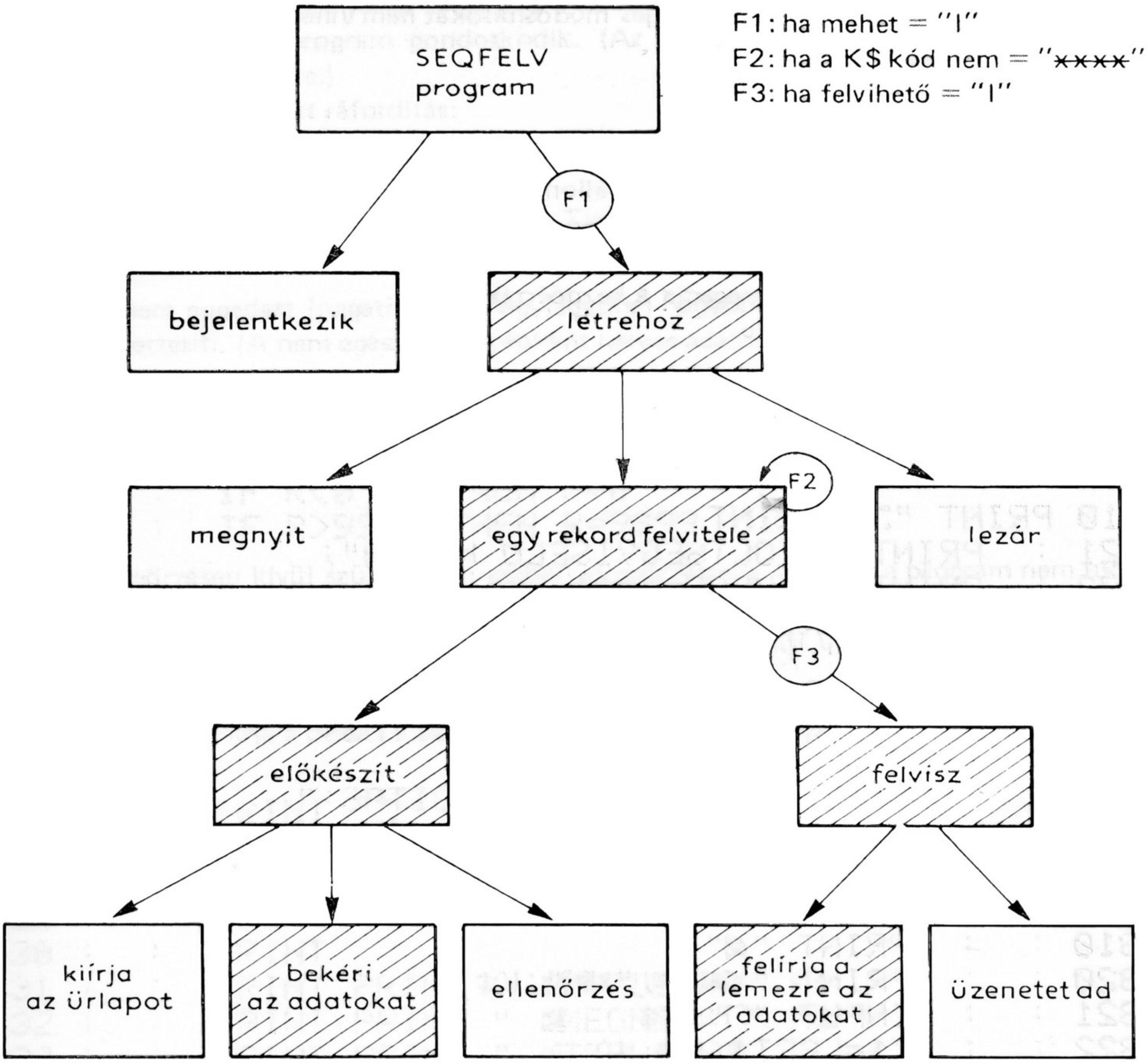
```
1 REM:
2 PRINT "♥"
3 PRINT
4 PRINT " KOLTSEGVISELOK FELVITELE"
5 PRINT " ----------------------------"
6 PRINT
8 S$=CHR$(13)
```

Bekéri a létrehozandó állomány A$ nevét (ami most KOLTSEGVISELOK):

```
10 PRINT
11 PRINT
12 INPUT " ▪ALLOMANY NEVE▪=";A$
13 PRINT
15 PRINT
```

Hibás névmegadás esetén a program még leállítható:

```
16 INPUT " MEHET=  I▪▪▪";V$
17 IF V$<>"I" THEN GOTO 999
```



11. ábra. Soros állomány létrehozása

A program feltételezi, hogy nem hibázunk; ha mégis, az igenlő választ kell bármilyen karakterrel felülbírálnunk.
Ha ezt nem tesszük, megnyitja az állományt:

```
20 OPEN 2,8,2,A$+",SEQ,WRITE"
```

A megnyitáshoz az OPEN utasítást használjuk. A paramétereinek jelentése: 2 = állományazonosító, 8 = egységszám, 2 = az adatcsatorna száma, A$ = az állomány neve (ez az, amit az INPUT kérdésre begépeltünk), SEQ = az állomány típusa (sequential, azaz soros), WRITE = az állomány megnyitási iránya (írás).
Megjegyezzük, hogy az állománykezelő információkat egyetlen karakteres jelsorozattá kell összekapcsolni, és az egyes részinformációkat egymástól vesszővel kell elválasztani. Jó, ha tudjuk, hogy a SEQ kulcsszó S, a WRITE pedig W betűvel rövidíthető – ugyanis az összekapcsolt jelsorozat mérete nem haladhatja meg a 40 karaktert.
(Természetesen az állomány nevét közvetlenül is megadhattuk volna:

```
20 OPEN 2,8,2,"KOLTSEGVISELOK,SEQ,WRITE"
```

Ez esetben azonban a később szükséges módosításokat nem vihetjük fel a lemezre ugyanezzel a felvivő programmal.)

FIGYELEM!
Emlékezzünk arra, hogy új állomány csak akkor nyitható meg létrehozásra (WRITE), ha még nem létezik, és az így megnyitott állományba csak írni (PRINT#) lehet. A lemezkezelő e szabályok betartását automatikusan ellenőrzi.

Ha a program eddig eljutott, megadja a kezdőértékeket:

```
30 K$=""
31 M$=""
32 T=0
33 R=0
```

Ezután a képernyőre kiír egy űrlapot, amely a megjelölt adatokkal töltendő ki:

```
110 PRINT "⌉":PRINT
121  :   PRINT " KOLTSEGVISELO KODJA";
122  :   PRINT " (VEGE=****):"
123  :   PRINT:PRINT:PRINT
131  :   PRINT " KOLTSEGVISELO MEGNEVEZESE:"
132  :   PRINT:PRINT:PRINT
141  :   PRINT " TERVEZETT RAFORDITAS:"
142  :   PRINT:PRINT:PRINT
151  :   PRINT " TENYLEGES RAFORDITAS:"
152  :   PRINT:PRINT:PRINT
```

Most a program az űrlap megfelelő rovatába viszi a kurzort, majd kéri a költségviselő K$ kódját; a begépelt válaszból csak az első négy karaktert figyelembe véve:

```
310  :   :   PRINT "▩"
320  :   :   PRINT "▩▩▩";"▮▮▮";K$
321  :   :   INPUT "⌉";K$
322  :   :   K$=LEFT$(K$,4)
```

Ha a K$ kód nem egy tényleges költségviselő kódja, azaz ha a végjelet gépeltük be, akkor befejezi a felvitelt:

```
323  :   :   IF K$="****" THEN GOTO 910
```

Végjelünk a "****". Lehetne például a "VEGE" szöveg is, de csak akkor, ha biztos, hogy nincs és nem is lehet ilyen kódú költségviselőnk. Még jobb, ha a program nem egy speciális kódtól áll le, hanem külön megkérdezi, hogy van-e még újabb költségviselő, és az igenlő vagy tagadó válasz értelmében jár el. Mi a legegyszerűbb megoldást választottuk, hogy ne vonjuk el a figyelmet az adatfelvitel algoritmusáról, és hogy hagyjunk önálló kísérletezési lehetőséget az olvasónak is.
Ha a program nem végjelet kapott, akkor azt létező kódnak tekinti, és bekéri a költségviselő megnevezését:

```
330  :   :   PRINT "▩▩▩";"▮▮▮";M$
331  :   :   INPUT "⌉";M$
332  :   :   M$=LEFT$(M$,30)
```

Természetesen az adatot az űrlap megfelelő rovatába kell begépelnünk; erről a kurzor alkalmas beállításával a program gondoskodik. (Az M$ megnevezésből csak az első 30 karaktert veszi figyelembe.)

Ezután jöhet a T tervezett ráfordítás:

```
340 :  :  PRINT "■■■■";"■■";T
341 :  :  INPUT "■";T
342 :  :  IF T<0    THEN T=0
343 :  :  IF T>9999 THEN T=9999
```

Itt a meg nem engedett (negatív vagy négyjegyűnél nagyobb) adatokat 0, illetve 9999 értékkel helyettesíti. (A nem egész, de egyébként helyes adatokat elfogadja.)

A program ugyanígy jár el a tényleges R ráfordítás bekérésénél is:

```
350 :  :  PRINT "■■■■";"■■";R
351 :  :  INPUT "■";R
352 :  :  IF R<0    THEN R=0
353 :  :  IF R>9999 THEN R=9999
```

A gépi ellenőrzésen kívül szükség van emberi ellenőrzésre is. Ugyanis a program nem tudhatja, hogy például a begépelt PTEV kód helyes-e, avagy inkább PTER-nek kellene lennie. Ugyanígy a téves adatokat a program önkényesen helyettesíti 0, illetve 9999 értékkel, amivel a gépkezelő nem feltétlenül ért egyet.

Ezért a program kiírja a bevett adatokat, olyan formában, ahogyan azok a lemezre fognak kerülni:

```
410 :  :  PRINT "■"
420 :  :  PRINT " EZ FOG A LEMEZRE KERULNI:"
421 :  :  PRINT " ------------------------------"
422 :  :  PRINT
430 :  :  PRINT
431 :  :  PRINT:PRINT " ■KOOD■ = ";K$
432 :  :  PRINT:PRINT " ■MEGN■ = ";M$
433 :  :  PRINT:PRINT " ■TERV■ =" ;T
434 :  :  PRINT:PRINT " ■TENY■ =" ;R
435 :  :  PRINT
```

Az adatokat begépelő személy most ellenőrizheti, hogy követett-e el gépelési hibát, avagy sem; és dönthet, hogy a rekord felvihető-e a lemezre, vagy ha kell, javítja az adatokat:

```
440 :  :  INPUT " ■FELVIHETO■ =I/N=";V$
441 :  :  IF V$="I" THEN GOTO 510
499 :  GOTO 110
```

Ha a gépkezelő úgy dönt, hogy az adatok között van hibás, a program újra adja az űrlapot, és újra kéri az adatokat.

Azonban válaszként eleve feltünteti a már begépelt értékeket, így a helyes adatokat nem kell újra begépelnünk, elég ha azonnal a RETURN gombot nyomjuk meg. A hibás adatok helyett sem kell elölről begépelni a hibátlan adatot, ugyanis a hibás begépelés a szokásos módon javítható.

Ha a rekordot a gépkezelő felvihetőnek minősítette, akkor azt a program felviszi a lemezre:

```
510  :  PRINT#2,K$;S$;M$;S$;T;S$;R
```

A rekord felírása a PRINT# utasítással történik. Ebben megadjuk az állományazonosítót, majd az azt követő vessző után, a lemezre írandó adatokat tartalmazó változók listáját, amely a K$, M$, T és R változókon kívül magába foglalja a *szeparátorjelet* tartalmazó S$ változót is. A felsorolt változóneveket pontosvesszővel választjuk el egymástól. Így az adatok közé kerülnek az S$-ban tárolt sorvége jelek; a rekord végére pedig ugyanilyen jel kerül (a RETURN miatt).

A felvitel után a program jelzi annak megtörténtét, a „FELVIVE" szöveg villogtatásával:

```
520  :  PRINT:PRINT:PRINT,
521  :  FOR I=1 TO 3
522  :  PRINT " FELVIVE";"■■■■■■■■■";
523  :  FOR J=1 TO 180 : NEXT J
524  :  PRINT " ■FELVIVE■";"■■■■■■■■■";
525  :  FOR J= 1 TO 180 : NEXT J
526  :  NEXT I
```

Ezután törli a begépelt adatokat, majd áttér az újabb rekord bekérésére:

```
530  :  K$=""
531  :  M$=""
532  :  T=0
533  :  R=0
599 GOTO 110
```

Ha nincs több felviendő rekordunk, azaz ha négy csillagot (végjelet) adtunk meg kódként, a program lezárja az állományt, és egy üzenettel kijelentkezik:

```
910 CLOSE 2
920 PRINT "□"
921 PRINT
922 PRINT " ";A$;" ALLOMANY"
923 PRINT
924 PRINT " *** KESZ ***"
```

Végül leáll:

```
999 END
```

FIGYELEM!

Az állomány lezárása a szokásos CLOSE utasítással történik. Ha elmulasztjuk, az állomány, ha fel is került a lemezre, *használhatatlan* lesz.

Ha idáig eljutottunk, mindenekelőtt mentsük ki a programot SEQFELV néven a SOROS lemezre.

Hagyjuk bent a lemezt, és adjunk RUN parancsot.

A program bejelentkezése után, az állomány nevének bekérésekor adjuk meg a KOLTSEGVISELOK nevet.

FIGYELEM!

Mielőtt a mehet kérdésre igenlő választ adnánk, ellenőrizzük, hogy a nevet hibátlanul gépeltük-e be, egyébként a későbbi programjaink az állományt nem fogják megtalálni.

Majd adatként az alábbiakat gépeljük be:

```
 KOLTSEGVISELO KODJA (VEGE=****):
? KTSGH.ALT.
 KOLTSEGVISELO MEGNEVEZESE:
? KOLTSEGHELYI ALTALANOS RAFORDITASOK
 TERVEZETT RAFORDITAS:
? -222
 TENYLEGES RAFORDITAS:
? 1234567
```

Amikor a program kiírja, hogy mi kerül a lemezre:

```
EZ FOG A LEMEZRE KERULNI:
-------------------------

[illegible] = KTSG
[illegible] = KOLTSEGHELYI ALTALANOS RAFORDI
[illegible] = 0
[illegible] = 9999
[illegible] =I/N=? N
```

adjunk „N'' választ, hiszen nem ezt akarjuk felvinni.

Ekkor visszakapjuk az űrlapot, és az adatokat javíthatjuk:

- A kódot hagyjuk úgy, ahogy van, mert megfelelő lesz számunkra így is. Egyszerűen nyomjuk meg a RETURN gombot.
- A megnevezést javítsuk ki; vigyük a kurzort a RAFORDI szöveg elejére, és írjuk felül azt szóközökkel. Ezután adjunk RETURN-t.
- Tervként gépeljük be a 200 értéket.
- A tény legyen nulla.

FIGYELEM!

Ha az új adat rövidebb, mint a régi, ne felejtsük el letörölni a „kilógó'' régi részeket.

Most már a javított rekordot kapjuk vissza. (Lásd a 86. oldalon!) Ekkor kiadhatjuk a felvihető kérdésre az "I" választ. Az első rekord tehát megvan. Vigyünk fel további rekordokat is. (Könnyebben ellenőrizhetjük a későbbi programok helyességét, ha a 86. oldalon bemutatott adatokat használjuk.) A program a KOOD=~~xxxx~~ válasz hatására áll le. Igyekezzünk az adatokat hiba nélkül begépelni. Ha mégis elírunk valamit, az első rekordnál bemutatott módon javíthatjuk.

EZ FOG A LEMEZRE KERULNI:

[illegible] = KTSG
[illegible] = KOLTSEGHELYI ALTALANOS
[illegible] = 200
[illegible] = 0
[illegible] =I/N=? I

EZ FOG A LEMEZRE KERULNI:

[illegible] = FELM
[illegible] = HELYZETFELMERES
[illegible] = 200
[illegible] = 100
[illegible] =I/N=? I

EZ FOG A LEMEZRE KERULNI:

[illegible] = ATAD
[illegible] = RENDSZER ATADASA
[illegible] = 50
[illegible] = 50
[illegible] =I/N=? I

EZ FOG A LEMEZRE KERULNI:

[illegible] = RTER
[illegible] = RENDSZERTERVEZES
[illegible] = 500
[illegible] = 800
[illegible] =I/N=? I

EZ FOG A LEMEZRE KERULNI:

[illegible] = PTER
[illegible] = PROGRAMTERVEZES
[illegible] = 600
[illegible] = 600
[illegible] =I/N=? I

EZ FOG A LEMEZRE KERULNI:

[illegible] = PKOD
[illegible] = PROGRAMKODOLAS
[illegible] = 300
[illegible] = 200
[illegible] =I/N=? I

EZ FOG A LEMEZRE KERULNI:

[illegible] = TEST
[illegible] = TESZTELES
[illegible] = 200
[illegible] = 0
[illegible] =I/N=? I

Megjegyezzük, hogy a lemezegységünk kissé furcsán fog a felvitelkor működni. A működést jelző piros lámpa a program indításakor kigyullad, és halljuk is, amint az író/olvasó fej dolgozik. A megnyitás hatására ugyanis megteszi a szükséges bejegyzéseket a tartalomjegyzékbe.

Az egyes rekordok felvitelekor azonban nem halljuk a lemezkezelés jellegzetes súrlódó zaját, viszont a program végén a négycsillagos végjel begépelése után ez legalább 2—3 másodpercig jól hallható.

A jelenség oka a *pufferkezelésben* rejlik. A kiírandó adatokat a lemezkezelő egy külön tárterületen, a *pufferban* gyűjti, és ténylegesen csak akkor írja fel a lemezre, ha a puffer megtelt.

Állományunk adatai alig valamivel haladják meg a 200 bájtot, így a puffer nem telik meg velük. Ezért nem történt írás a felvitelkor.

Az állomány lezárásakor azonban a puffer mindenképpen felíródik a lemezre, akár megtelt, akár nem. Csak ezután módosítja a lemezkezelő a lemeztérképet. Ez történt a mi felvivő programunk futásakor is.

Mellesleg figyeljük meg azt is, hogyan használja ki a program az interaktivitásban rejlő lehetőségeket. Itt elsősorban nem arról van szó, hogy billentyűzetről kéri be az adatokat, hanem arról, ahogyan ezt teszi; azaz, hogy a futása közben információkat ad ki a gépkezelőnek, például arról, hogy mi kerül a lemezre, és a gépkezelő ezen információk birtokában dönthet, hogy mi legyen a végrehajtás további menete, a döntést közölheti a programmal — ,,aki'' azután ennek megfelelően jár el.

FELDOLGOZÁS

Van tehát állományunk, most már feldolgozhatjuk.

A soros feldolgozás lényege, hogy sorra vesszük az állomány rekordjait, azaz beolvassuk őket a lemezről, majd a rekordok adataival elvégezzük a feldolgozáshoz szükséges műveleteket, és végül az eredményt közvetlenül felhasználjuk, például kiírjuk, vagy későbbi felhasználás céljából megőrizzük, mondjuk egy másik állományban. Ezt mindaddig folytatjuk, amíg el nem érjük az állomány végét (12. ábra).

Az ábra szerinti feltételek a következők:

F1: ha mehet = "I"

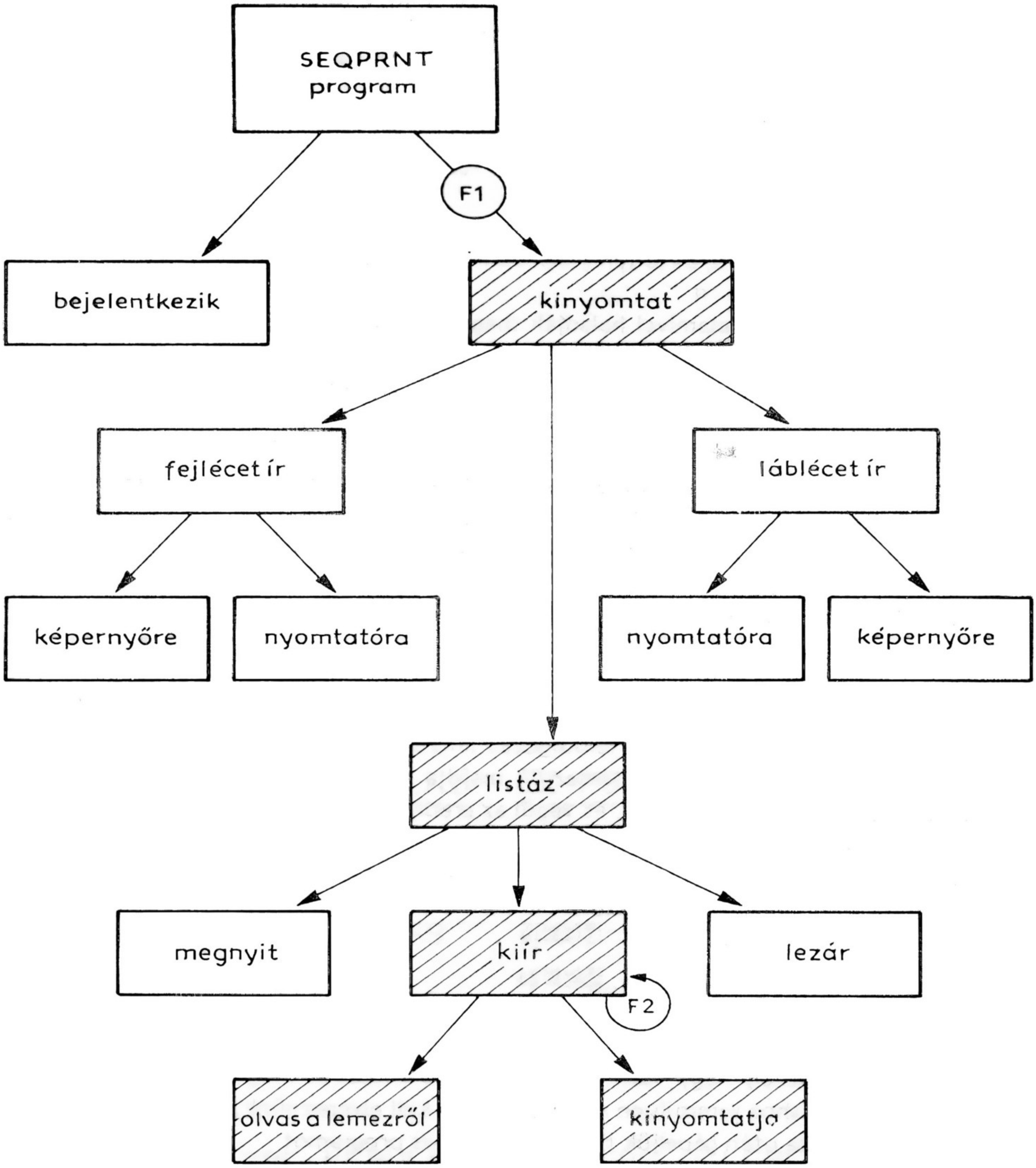
F2: ha van még adat, azaz STATUS=0

Vegyük a legegyszerűbb esetet, amikor az állomány adatait kinyomtatjuk. Egyszerűsítsük a feladatot azzal, hogy csak a költségviselők kódját és megnevezését nyomtassuk ki. (Így a feldolgozáshoz semmilyen műveletet nem kell elvégeznünk.)

NEW parancs kiadása után gépeljük be az alábbi teljes programot.

A program bejelentkezik, és vár arra, hogy elindítsuk vagy leállítsuk:

```
1 PRINT "♥"
2 PRINT
3 PRINT " KOLTSEGVISELOK NYOMTATASA"
4 PRINT " -------------------------"
5 PRINT
6 PRINT
7 INPUT " MEHET =I/N=";V$
8 IF V$<>"I" THEN GOTO 999
```



12. ábra. Soros állomány kinyomtatása

A képernyőn jelzi, hogy a listát a 4-es nyomtatón fogjuk létrehozni, majd el is készíti az egyszerű fejlécet:

```
110 PRINT
120 PRINT,"■LISTA A 4-ES NYOMTATON■"
200 REM ------------------------------------------------ FEJLECIRAS
210 OPEN 4,4
220 PRINT#4
230 PRINT#4," KOLTSEGVISELOK LISTAJA"
```

```
240 PRINT#4," ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
250 PRINT#4
260 PRINT#4," KOD:    |  MEGNEVEZES:"
270 PRINT#4," ---------+-------------------------------------------------
```

Megnyitja a lemezállományt:

```
310 OPEN 2,8,2,"KOLTSEGVISELOK,SEQ,READ"
```

FIGYELEM!

Emlékezzünk arra, hogy csak olyan állományt lehet olvasásra (READ) megnyitni, amely már létezik a lemezen. Az így megnyitott állományra csak INPUT# utasítás adható ki, PRINT# nem. Ha az állományt nem a megnyitással összhangban használjuk, hibaállapot következik be, ami nem feltétlenül jár programmegszakítással.

A megnyitás után a program beolvas egy rekordot, és egy VEGE változóba kimenti az állomány végét jelző STATUS értéket:

```
410 INPUT#2,K$,M$,T,R
420 :  VEGE=STATUS
```

Az I. részben már írtuk, hogy a lemezkezelő fenntart egy speciális változót, a STATUS-t az állomány végének a jelzésére. Ennek a tartalma 0, ha az olvasás nem érte el az állomány végét, és 64, ha elérte. Minden más érték valamilyen, általában olvasási hibára utal.

A program ezután kinyomtatja a beolvasott adatok közül a K$ kódot és az M$ megnevezést:

```
430 :  PRINT#4
440 :  PRINT#4," ";K$;" --- ";M$
```

Megvizsgálja a VEGE jelet, hogy van-e még adat az állományban. Ha van, azaz a STATUS az olvasáskor még nem állt át nulláról 64-re, akkor visszatér a következő rekord olvasására:

```
499 IF VEGE=0 THEN GOTO 410
```

Ha viszont nincs több adat, akkor lezárja a lemezállományt:

```
510 CLOSE 2
```

Majd elkészíti a lista láblécét:

```
610 PRINT#4
620 PRINT#4," ----------------------------------------------------------"
630 PRINT#4
640 CLOSE 4
```

Ennek befejeztével üzen a képernyőn, hogy a lista elkészült:

```
710 PRINT
720 PRINT,"■LISTAZAS KESZ■"
```

Végül leáll:

```
999 END
```

Ezt a programot is mentsük ki a SOROS lemezünkre, adjuk neki a SEQPRINT nevet. Most pedig jöhet a RUN parancs. Ha nem vétettünk gépelési hibát, a program az alábbi listát fogja elkészíteni:

```
KOLTSEGVISELOK LISTAJA
▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒

KOD:    |  MEGNEVEZES:
--------+----------------------------------------
KTSG --- KOLTSEGHELYI ALTALANOS

FELM --- HELYZETFELMERES

RTER --- RENDSZERTERVEZES

PTER --- PROGRAMTERVEZES

PKOD --- PROGRAMKODOLAS

TEST --- TESZTELES

ATAD --- RENDSZER ATADASA
--------+----------------------------------------
```

A teljes program így néz ki:

```
1 PRINT "▫"
2 PRINT
3 PRINT " KOLTSEGVISELOK NYOMTATASA"
4 PRINT " --------------------------"
5 PRINT
6 PRINT
7 INPUT " MEHET =I/N=";V$
8 IF V$<>"I" THEN GOTO 999
100 REM ------------------------------------ ELOKESZITES
110 PRINT
120 PRINT,"▪LISTA A 4-ES NYOMTATON■"
200 REM ------------------------------------ FEJLECIRAS
210 OPEN 4,4
220 PRINT#4
230 PRINT#4," KOLTSEGVISELOK LISTAJA"
240 PRINT#4," ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
250 PRINT#4
260 PRINT#4," KOD:    |  MEGNEVEZES:"
270 PRINT#4," --------+----------------------------------------"
310 OPEN 2,8,2,"KOLTSEGVISELOK,SEQ,READ"
400 REM ------------------------------------ NYOMTATAS
410 INPUT#2,K$,M$,T,R
420 :  VEGE=STATUS
430 :  PRINT#4
440 :  PRINT#4," ";K$;" --- ";M$
499 IF VEGE=0 THEN GOTO 410
500 REM ------------------------------------ LABLECIRAS
```

```
510 CLOSE 2
610 PRINT#4
620 PRINT#4," ──────────┴──────────────────────────────"
630 PRINT#4
640 CLOSE 4
700 REM ──────────────────────────────────── BEFEJEZES
710 PRINT
720 PRINT,"■LISTAZAS KESZ■"
999 LOAD "SEQREND",8
```

Megjegyezzük, hogy a programban *hátultesztelő* ciklust szerveztünk, vagyis feltételezzük, hogy az állomány legalább egy rekordot tartalmaz.

Ha ez nem garantált, át kell szerveznünk a feldolgozás menetét. Ehhez azonban figyelembe kell vennünk a STATUS változó sajátosságait.

A lemezkezelő ugyanis mindig csak egyetlen STATUS változót tart fenn, akárhány állományt is használunk, és annak értéke mindig a legutoljára kezelt állományra vonatkozó információt tartalmazza.

Ezért volt szükségünk a VEGE változóra, ezért kellett oda kimentenünk a STATUS tartalmát a 410 INPUT utasítás után. Ha ugyanis a 499 IF utasításban közvetlenül a STATUS tartalmát vizsgálnánk meg, a programunk végtelen ciklusba esne, mivel a STATUS mindig nulla lesz, ha a 440 PRINT utasítás sikeresen végrehajtódott. A kimentéssel azonban elértük, hogy a megvizsgáláskor nem a STATUS aktuális értékét vesszük figyelembe, hahanem azt, amivel a kimentéskor rendelkezett. Így a példánkban nem a 4-es, hanem a 2-es állomány állapotát vizsgáljuk.

FIGYELEM!

Az állomány végét jelző 64-es értéket a STATUS változó *nem akkor veszi fel, amikor az állomány vége után olvasunk, hanem már az utolsó adat beolvasásakor.*

Ezért nem léphetünk ki a ciklusból a 420-as sorban. Ha megtennénk, „elveszítenénk" az utolsó rekordot; az utolsó költségviselő adatai nem jelennének meg a listán. A programunk azonban a STATUS átváltásakor éppen beolvasott adatokat még feldolgozza.

FIGYELEM!

Ha az állomány végének elérése után újabb INPUT# utasítást adunk ki, a STATUS ismét felveszi a 64-es értéket, az olvasás pedig nem hajtódik végre, és az INPUT# utasításban felsorolt változók megtartják korábbi értéküket.

Végül megjegyezzük, hogy a *lemezkezelő az olvasást is pufferezve hajtja végre.* Vagyis az első INPUT# hatására beolvas egy teljes blokkot, annak tartalmával feltölti a puffert, majd innen adogatja az értékeket az INPUT# utasításokban szereplő változóknak. Újabb blokkot csak akkor olvas, ha valamely INPUT# utasítást már nem tud a pufferből adattal kielégíteni.

Alighogy elkészültünk vele, máris javasoljuk a program módosítását. Láthatjuk ugyanis, hogy ebben a programban nem kértük be az állomány nevét, hiszen a program kizárólag a KOLTSEGVISELOK állomány kilistázására készült, és a program az általa használt rekordszerkezet miatt másra nem is alkalmas.

Azonban előfordulhat — amint később látni fogjuk —, hogy egy lemezen a KOLTSEGVISELOK állományunk több változatban is létezhet, persze mindegyikük más-más néven.

Ezért célszerű a programot úgy módosítani, hogy ezeket a változatokat is listázni tudja, azaz hogy bekérje az állomány nevét.

```
7 INPUT" ALLOMANY=  -■■■■";A$
8 A$=LEFT$(A$,16)
9 IF LEFT$(A$,1)="-" THEN GOTO 999
310 OPEN 2,8,2,A$+",SEQ,READ"
```

BŐVÍTÉS

Előfordulhat, hogy a meglevő állományunkba újabb költségviselőt kell felvinnünk – vagy azért, mert a létrehozáskor kifelejtettük, vagy mert azóta keletkezett.

Biztosítanunk kell tehát a beszúrás lehetőségét, amit úgy oldhatunk meg, hogy az új rekordokat az állomány végére vagy elejére visszük fel. Rendezetlen állományoknál nem okoz problémát, hogy ezáltal „még rendezetlenebbé" válnak.

A legegyszerűbb megoldás nyilvánvalóan az, hogy a bővítést az állomány továbbírásra való megnyitásával hajtjuk végre. Ekkor egy alkalmas felvivő programmal, ilyenünk pedig már van, ott folytathatjuk az állomány feltöltését, ahol az előző felvitel befejezésekor abbahagytuk.

A továbbírásra az állományunkat a

```
20 OPEN 2,8,2,"KOLTSEGVISELOK,SEQ,APPEND"
```

utasítással nyithatjuk meg.

Persze ne ezt az utasítást tegyük be a SEQFELV programunkba, hanem úgy módosítsuk azt, hogy megkérdezze, milyen megnyitást kívánunk, és a válasznak megfelelően járjon el:

```
20 PRINT
21 INPUT" UJ/FOLYT =";I$:I$=LEFT$(I$,1)
22 IF I$="U" THEN I$="W":GOTO 25
23 IF I$="F" THEN I$="A":GOTO 25
24 GOTO 21
25 OPEN 2,8,2,"KOLTSEGVISELOK,S,"+I$
```

Ezzel a bővítést egyszerűen megoldottuk. Ki is próbálhatjuk. A végrehajtás sikerességét a SEQPRNT programmal ellenőrizhetjük:

```
KOLTSEGVISELOK LISTAJA

KOD:   | MEGNEVEZES:
-------+--------------------------------

KTSG --- KOLTSEGHELYI ALTALANOS

FELM --- HELYZETFELMERES

RTER --- RENDSZERTERVEZES

PTER --- PROGRAMTERVEZES
```

```
PKOD --- PROGRAMKODOLAS

TEST --- TESZTELES

ATAD --- RENDSZER ATADASA

MODO --- MODOSITAS
```

A módszer egyetlen veszélye, hogy a sikertelen bővítés esetén maga a bővítendő állomány is megsérülhet. Ezért csak megbízható hardver esetén célszerű használni.

MÁSOLÁS

Növelhetjük a biztonságot, ha az állományunkról a bővítés előtt egy tartalék példányt készítünk:

```
OPEN 15,8,15

READY.
PRINT#15,"COPY:TARTALEK=KOLTSEGVISELOK"

READY.
CLOSE 15

READY.
```

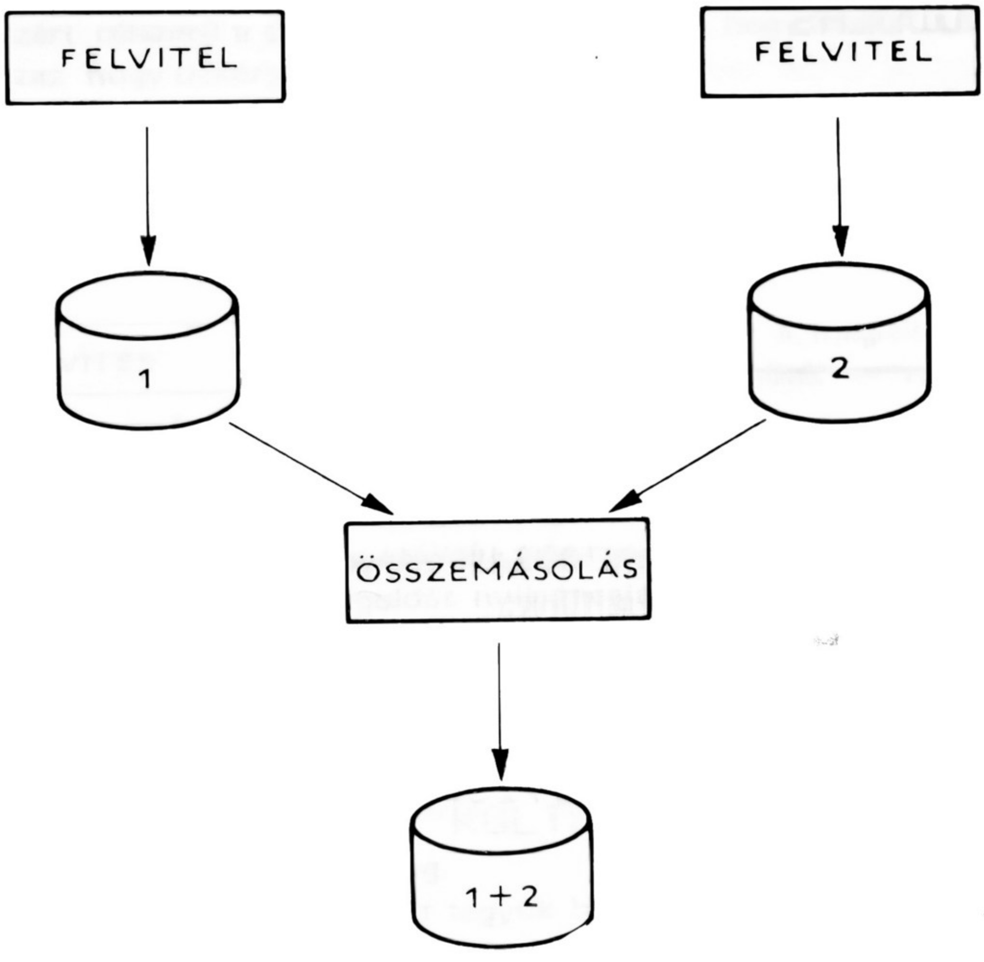
Javasoljuk is, hogy ezt tegyük meg. Így később akármit tehetünk a KOLTSEGVISELOK állományunkkal, akárhogyan módosítjuk is, mindig visszatérhetünk az eredeti, kiinduló állományhoz – anélkül, hogy azt újra fel kellene vinnünk.

Természetesen nem kell minden állományról mindig másolatot készíteni. Külön szervezési kérdés, hogy melyik állományról, annak mikori állapotáról, a feldolgozás mely fázisában készüljön kimentés. Mindezeket a feldolgozás és a környezet ismeretében kell meghatározni, azzal együtt, hogy az archív állományok meddig őrizendők meg.
Az előny nyilvánvaló: az állomány esetleges megsemmisülése, megsérülése esetén nem kell mindent elölről kezdeni. Megjegyezzük, hogy ez nem is mindig lehetséges.

ÖSSZEMÁSOLÁS

Ha már a COPY funkciónál tartunk, hadd emlékeztessük az olvasót az I. részben már ismertetett tényre: a COPY lemezkezelő paranccsal soros állományok, legfeljebb négy állományig, összekapcsolhatók.
Ezt a tulajdonságot kihasználhatjuk az állomány bővítésére (13. ábra).

Akár ki is próbálhatjuk:
- Hozzunk létre a SEQFELV programmal egy BOVITES állományt, a meglevőtől eltérő adatokkal.
- Ellenőrzésképpen a SEQPRNT programmal készítsünk róla egy listát.
- Majd a BOVITES állományt kapcsoljuk hozzá a KOLTSEGVISELOK állományhoz.



13. ábra. Az összemásolás menete

```
KOLTSEGVISELOK LISTAJA

KOD:     |  MEGNEVEZES:
---------+--------------------------------

DOKU --- DOKUMENTALAS

GARA --- GARANCIALIS JAVITAS

---------+--------------------------------
```

```
OPEN 15,8,15

READY.
PRINT#15,"COPY:UJ=KOLTSEGVISELOK,BOVITES"

READY.
CLOSE 15

READY.
```

Ekkor létrejön az UJ állomány, amely a két korábbiból áll, a megadott sorrendben összekapcsolva. Erről a SEQPRNT programmal győződhetünk meg:

```
KOLTSEGVISELOK LISTAJA

KOD:   | MEGNEVEZES:
-------+----------------------------------

KTSG --- KOLTSEGHELYI ALTALANOS

FELM --- HELYZETFELMERES

RTER --- RENDSZERTERVEZES

PTER --- PROGRAMTERVEZES

PKOD --- PROGRAMKODOLAS

TEST --- TESZTELES

ATAD --- RENDSZER ATADASA

MODO --- MODOSITAS

DOKU --- DOKUMENTALAS

GARA --- GARANCIALIS JAVITAS

-------+----------------------------------
```

Az ilyen bővítésre önálló programot is írhatunk:

```
110 PRINT"♥":PRINT
120 PRINT" ALLOMANY BOVITESE"
130 PRINT:PRINT
210 PRINT:INPUT" BOVITENDO = ";E$
220 PRINT:INPUT" BOVITO    = ";M$
230 PRINT:INPUT" BOVITETT  = ";U$
310 OPEN 15,8,15
320 PRINT#15,"COPY:"+U$+"="+E$+","+M$
330 CLOSE 15
910 PRINT:PRINT" *KESZ*"
999 END
```

Ha meg akarjuk őrizni, mentsük ki SEQBOVT néven.

Egyébként az összemásolás az eredeti állományokat nem változtatja, és nem semmisíti

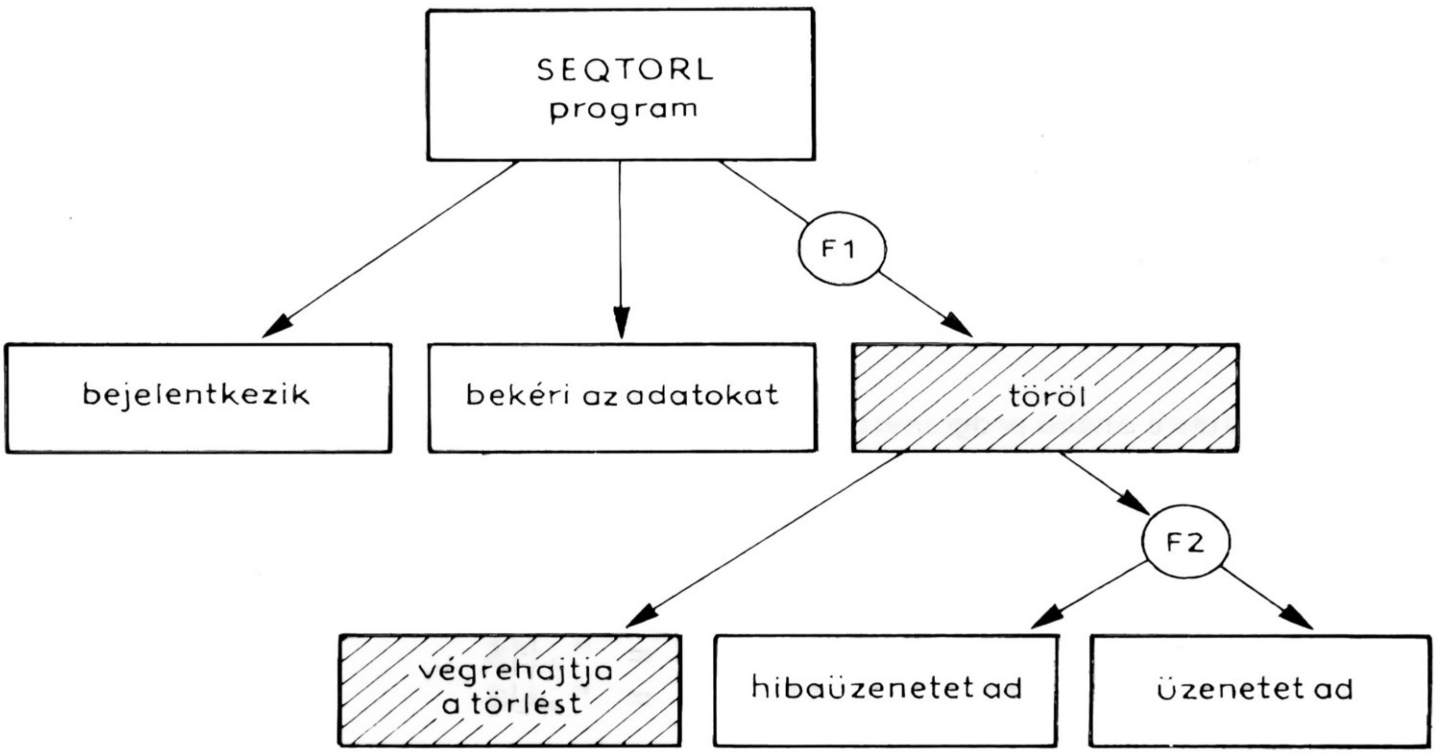
meg — az új állomány valóban másolás útján készül, és a másolatok, nem pedig az eredeti állományok összekapcsolásából áll.

Megjegyezzük, hogy kezdetben a hagyományos, **azaz nagygépes kötegelt adatfeldolgozás**ban a soros állományokat általában összemásolással **bővítették**, persze ezt programból kellett végrehajtani; végig kellett olvasni és átírni az egyik állományt, majd ugyanezt tenni a másikkal. A kezdeti fájlkezelő rendszerek nemigen tettek más megoldást lehetővé. A továbbírásra való megnyitás és a COPY vagy azzal ekvivalens funkció viszonylag újabb kori szolgáltatás.

TÖRLÉS

Kérjük le a SOROS lemez tartalomjegyzékét! Láthatjuk, hogy most a költségviselőkkel kapcsolatban 4 állományunk van: KOLTSEGVISELOK, amely egy elavult, bővítetlen állomány; TARTALEK, amely a fentinek egy másolata; UJ, amely a jelenlegi, bővített állapotot tükrözi; és BOVITES, amely már felesleges, mert beépült az UJ-ba.

Jól szervezett feldolgozás esetén kevésbé, rosszul szervezettnél fokozottabban, teszteléskor viszont mindig fennáll az a veszély, hogy ,,ellepnek'' bennünket az állományok.
Szükséges tehát valamilyen rendet tartanunk, és a feleslegessé vált állományokat rendszeresen törölnünk. Ez a SCRATCH funkcióval könnyen végrehajtható.
Kulturáltabban és kényelmesebben megvalósítható a törlés, ha nem alkalmi parancsokkal végezzük el, hanem programot írunk rá:



14. ábra. Állomány törlése

Az ábra alapján:
- F1: ha törölhető = ''I''
- F2: ha a H hibakód nagyobb mint 19

A program bejelentkezik:

```
1 PRINT "□"
2 PRINT
3 PRINT " ALLOMANYOK TORLESE"
4 PRINT " ------------------"
5 PRINT
6 PRINT
```

Bekéri a törlendő állomány A$ nevét:

```
110 PRINT " ADJA MEG ANNAK AZ ALLOMANYNAK A NEVET,"
120 PRINT "           ▄▄▄▄▄▄▄▄▄▄▄          "
130 PRINT " AMELYET   ■TOROLNI■   KIVAN!"
140 PRINT "           ▔▔▔▔▔▔▔▔▔▔▔          "
150 PRINT
160 INPUT " *TORLENDO=";A$
170 PRINT
180 PRINT " ----------   ■";A$;"■ ALLOMANY"
190 PRINT
```

Nagyon vigyáznunk kell, nehogy tévedésből olyan állományt töröljünk, amire még szükség lehet. Itt még ellenőrizhetjük, hogy a megfelelő nevet gépeltük-e be, és szükség esetén letilthatjuk a törlést:

```
200 INPUT " TOROLHETO=  N■■■■";V$
210 IF V$<>"I" THEN GOTO 999
```

Ha igenlő, azaz V$="I" választ adtunk, a program végrehajtja a törlést:

```
220 OPEN 15,8,15
230 PRINT#15,"SCRATCH:"+A$
```

Közben lekérdezi a parancs-csatornát:

```
240 INPUT#15,H,H$
250 CLOSE 15
```

Ha nem kapott H hibajelzést, üzenetet ad a törlésről:

```
260 IF H>19 THEN GOTO 310
270 PRINT
280 PRINT " ■";A$;"■ TOROLVE"
290 PRINT
300 GOTO 999
```

Sikertelen törlés esetén közli velünk a parancs-csatorna H$ hibaüzenetét:

```
310 PRINT
320 PRINT " A TORLES SIKERTELEN VOLT!"
330 PRINT
340 PRINT " OKA: ";H$
350 PRINT
```

Végül mindkét esetben leáll:

```
999 END
```

Gépeljük be ezt a programot is, persze NEW után, és mentsük ki a lemezünkre SEQTORL néven.

Ezután kipróbálhatjuk.

Megjegyezzük, hogy a programmal nemcsak soros adatállományok, hanem programok és relatív állományok is törölhetők, de esetenként mindig csak egy. Azonban két utasítással:

```
199 IF A$="-" THEN GOTO 999
399 GOTO 1
```

könnyen módosíthatjuk úgy, hogy mindaddig kérje a törlendő állományokat, és mindaddig törölje azokat, amíg a törlendő neveként egy mínuszjelet nem gépelünk be. Ilyenkor módosítanunk kell a 210-es sort is:

```
210 IF V$<>"I" THEN GOTO 1
```

FIGYELEM!

Sikeres törlés esetén a parancs-csatornáról nem a szokásos 0 hibakód és OK üzenet jön be, hanem hibakód = 1 és hibaüzenet = FILES SCRATCHED értékeket kapunk. Ez tehát *nem jelent hibát.* De ugyanezt a visszajelzést kapjuk, ha a törlendő állomány már nincs a lemezen — tehát ténylegesen nem is hajtódott végre törlés.

Töröljük a programmal a BOVITES és a KOLTSEGVISELOK állományokat, valamint a TARTALEK-ot is. Például így:

```
ALLOMANYOK TORLESE
------------------

ADJA MEG ANNAK AZ ALLOMANYNAK A NEVET,
AMELYET TOROLNI KIVAN!

*TORLENDO=? KOLTSEGVISELOK
----------           KOLTSEGVISELOK ALLOMANY
TOROLHETO=? I
A TORLES SIKERTELEN VOLT!
OKA: DRIVE NOT READY


ALLOMANYOK TORLESE
------------------

ADJA MEG ANNAK AZ ALLOMANYNAK A NEVET,
AMELYET TOROLNI KIVAN!

*TORLENDO=? KOLTSEGVISELOK
----------           KOLTSEGVISELOK ALLOMANY
TOROLHETO=? I
KOLTSEGVISELOK TOROLVE
```

Ezek után, ha túl óvatosak vagyunk, készítsünk az UJ állományról egy TARTALEK másolatot. Ez most, a tesztelési fázisban nem feltétlenül szükséges. Ha meg akarjuk tartani az eredeti TARTALEK-ot, ne töröljük – de ekkor az újabb másolatot nem hozhatjuk létre ugyanezen a néven.

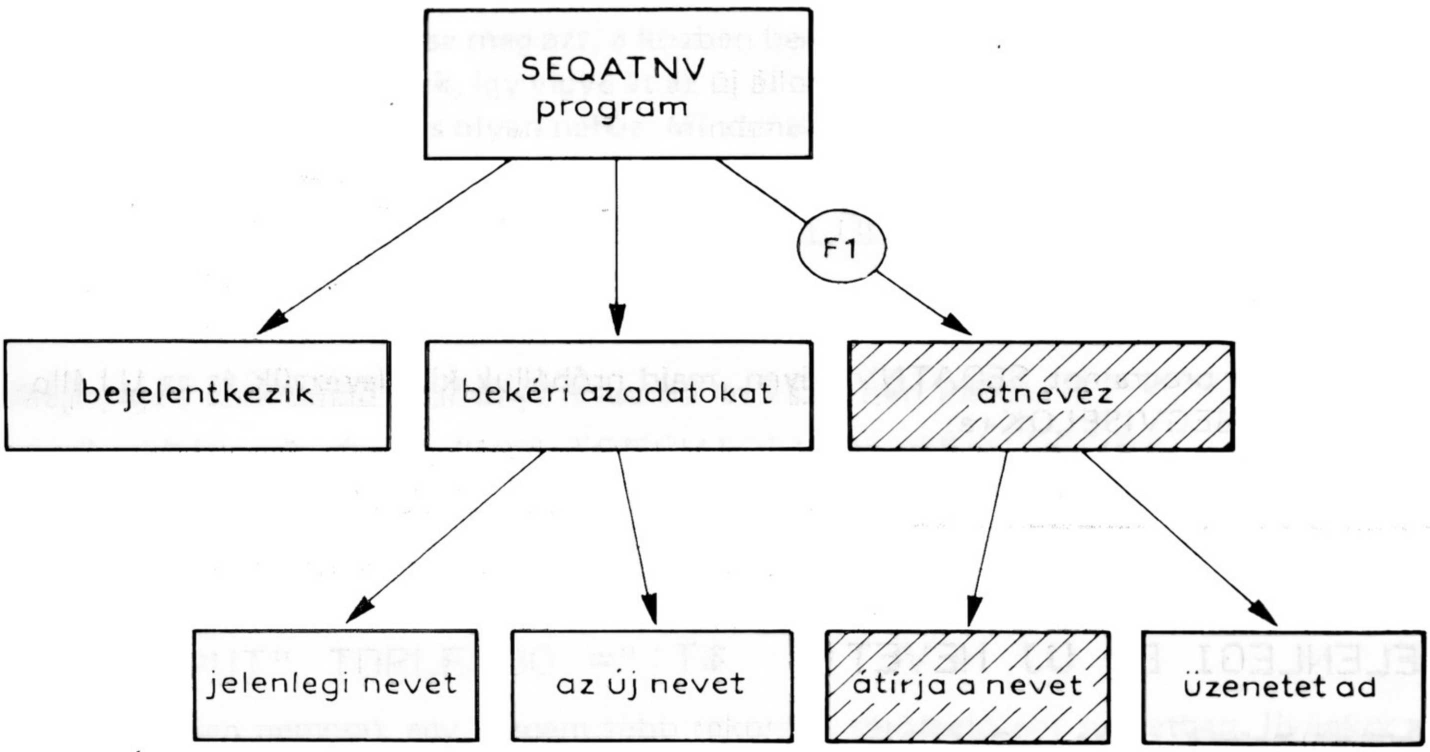
ÁTNEVEZÉS

A költségviselőink aktuális adatai, eltekintve az esetleges másolattól, most az UJ állományban vannak. Zavaró azonban, hogy az állomány neve nem KOLTSEGVISELOK. Ez nem pusztán esztétikai kérdés. Fontos, hogy ahogyan az állományokba, úgy a nevükbe se „fulladjunk bele", különben nem tudunk rendet tartani.

Mellesleg egy több programból álló adatfeldolgozó rendszer bizonyos programjai megkövetelhetik, hogy a feldolgozandó állományoknak meghatározott nevük legyen. (Ilyen volt eredetileg a SEQPRNT is, amit pusztán a tesztelés kedvéért változtattunk meg.)

Az ilyen programok többnyire nem azért használnak kötött állománynevet, mert lusták vagyunk beépíteni a név bekérését, hanem mert a kötött név biztonságosabbá teszi a feldolgozást; például lehetővé teszi, hogy a program ellenőrizhesse, hogy az az állomány van-e a lemezen, amelyet fel kell dolgoznia, vagy hogy egy adott állományhoz a hozzáférés jogosult-e stb. Tetszés szerint megadható állománynevek esetén ezek az ellenőrzések megkerülhetők.

Az állomány neve a RENAME funkcióval könnyen és gyorsan átírható, de miként a törlésnél, célszerű az átnevezésre is programot írni:



15. ábra. Állomány átnevezése

Ezt is gépeljük be, NEW parancs után.

A program bejelentkezik:

```
1 PRINT "♥"
2 PRINT
3 PRINT " ALLOMANY ATNEVEZESE"
4 PRINT " ___________________"
5 PRINT
6 PRINT
```

Bekéri az állományneveket:

```
110 PRINT " KEREM ADJA MEG AZ ALLOMANY"
120 PRINT " JELENLEGI ES UJ NEVET!"
130 PRINT
140 PRINT " ___________        "
150 INPUT " ▚JELENLEGI■ NEV =";JN$
160 PRINT " ‾‾‾‾‾‾‾‾‾‾‾        "
170 PRINT
180 PRINT " ___________        "
190 INPUT " ▚ * UJ *  ■ NEV =";UN$
200 PRINT " ‾‾‾‾‾‾‾‾‾‾‾        "
210 PRINT
```

Ellenőrzési lehetőséget ad:

```
220 PRINT
230 INPUT " MEHET =  N■■■";V$
240 IF V$<>"I" THEN GOTO 999
```

Végrehajtja az átnevezést:

```
310 OPEN 15,8,15
320 PRINT#15,"R:"+UN$+"="+JN$
330 CLOSE 15
```

Üzenetet ad az úi állománynévről:

```
910 PRINT
920 PRINT " AZ ALLOMANY NEVE ▚";UN$;"■"
```

Leáll:

```
999 END
```

Mentsük ki a programot SEQATNV néven, majd próbáljuk ki. Nevezzük át az UJ állományt KOLTSEGVISELOK-re:

```
ALLOMANY ATNEVEZESE
-------------------


KEREM ADJA MEG AZ ALLOMANY
JELENLEGI ES UJ NEVET!

JELENLEGI NEV =? UJ

 * UJ *   NEV =? KOLTSEGVISELOK

MEHET =? I
AZ ALLOMANY NEVE KOLTSEGVISELOK
```

Megjegyezzük, hogy a programmal nemcsak soros állományok, hanem programok és relatív állományok neve is megváltoztatható.

FIGYELEM!
A program nincs felkészítve a hibák kezelésére, így ügyeljünk arra, hogy a régi név szerepeljen a lemezen, az új név pedig ne létező állomány neve legyen.

A hibafigyelés és a törlésnél látott ismétlési lehetőség természetesen beépíthető lenne a programba. Nem tettük meg, hogy ne vegyük el az olvasó kedvét az önálló munkától.

SZŰKÍTÉS

A költségviselőkkel természetesen az is előfordulhat, hogy némelyikük megszűnik, vagy valamelyiküket tévedésből vittük fel az állományba. Ilyenkor ezeket törölni kell. Ezt nevezzük az állomány szűkítésének.

A törlés lényege, hogy végigolvasva az állományt, mindazon rekordjait átmásoljuk egy új (szűkített) állományba, amelyeket meg kívánunk tartani.
Az alábbiakban egy olyan törlőprogramot mutatunk be, amely kihasználja az interaktivitás lehetőségeit, azaz a törlés mindvégig a gépkezelő (operátor) felügyelete alatt folyik.
Erre alapvetően két lehetőség van. Az egyik az, hogy a program sorban olvassa a rekordokat, és mindegyikről megkérdezi, hogy törölhető-e, avagy sem. Ha nem, felviszi az új állományba; ha törölhető, akkor nem viszi fel. Ezt a módszert csak nagyon kis állományoknál érdemes választani.
Sokkal életrevalóbb az az ötlet, hogy a program kérdezze meg a törlendő rekord azonosítóját, majd önállóan keresse meg azt, a közben beolvasott rekordokat pedig tekintse automatikusan nem törlendőnek, így vigye át az új állományba.
Ilyen programot írni nem is olyan nehéz. Mindenekelőtt bejelentkezik:

```
110 PRINT"♥":PRINT
120 PRINT" KOLTSEGVISELOK ALLOMANY SZUKITESE"
130 PRINT:PRINT
140 S$=CHR$(13)
```

Megnyitja a szűkítendő állományt olvasásra és a szűkített írásra:

```
210 OPEN 2,8,2,"KOLTSEGVISELOK,SEQ,READ"
220 OPEN 3,8,3,"SZUKITETT,SEQ,WRITE"
```

Bekéri az operátortól a törlendő költségviselő T$ kódját:

```
310 PRINT
320 INPUT" TORLENDO =";T$
```

Természetesen nemcsak egy, hanem több rekord is törölhető egy menetben. Ilyenkor az operátornak a kódokat előfordulásuk sorrendjében kell megadnia. Ez nem túl erős megkötés, hiszen az operátor nem fejből javít, hanem egy, a kezében levő, és előzőleg megfelelően megjelölt lista alapján.
Mellesleg az se baj, ha téved, és kifelejt egy törlendőt, mivel a szűkítés ismételten végrehajtható, és akkor a kimaradt rekord törölhető. Ugyanez a helyzet, ha hibás kódot gépel be az operátor. Ezt a program nem fogja megtalálni, és ilyenkor semmit sem töröl.
A törlést leállítani éppen ezért úgy lehet, hogy nemlétező kódot, mondjuk négy kötőjelet ("––––") adunk meg.

Ha a program megkapta a törlendő rekord T$ azonosítóját, beolvas egy rekordot a szűkítendő állományból:

```
410 INPUT#2,K$,M$,T,R
```

Külön V jelzőbe kimenti az állományvége (STATUS) jelet, és törli a törlést engedélyező vagy tiltó V$ jelet:

```
420 V=STATUS:V$=""
```

Ha a beolvasott K$ rekord nem azonos a T$ törlendővel, akkor változtatás nélkül átviszi a szűkített állományba, és újabb rekordot olvas be:

```
430 IF K$<>T$ THEN PRINT#3,K$;S$;M$;S$;T;S$;R:GOTO 510
```

Ha viszont a költségviselő K$ kódja megegyezik a begépelt T$ törlendő kóddal, rákérdez az operátorra, hogy valóban ezt akarta-e törölni.
Nemleges V$ válasz esetén felírja a rekordot a szűkített állományba. Igenlő V$ válasz esetén nem viszi át a rekordot. Bármilyen más V$ válasz esetén újra kérdez.

FIGYELEM!
A törlés veszélyes funkció. Ugyan a tévedésből törölt rekord egy bővítő menettel egyszerűen hozzáírható az állományhoz, de csak akkor, ha az adatai az állományon kívül, például listán, megvannak, és hozzáférhetők. A program ezért kérdez rá külön minden egyes törlésre.

```
440 PRINT"";TAB(18);
450 INPUT" TOROLHETO =I/N=";V$
460 IF V$="N" THEN PRINT#3,K$;S$;M$;S$;T;S$;R:GOTO 510
470 IF V$="I" THEN GOTO 510
480 GOTO 440
```

Ha ekkor még nem érte el a szűkítendő állomány végét, új T$ törlendőt kér, vagy új rekordot olvas — attól függően, hogy talált vagy nem talált törlendő rekordot:

```
510 IF V=0 AND V$<>"" THEN GOTO 310
520 IF V=0 AND V$= "" THEN GOTO 410
```

A szűkítendő állomány végének elérésekor mindkét állományt lezárja:

```
610 CLOSE 3
620 CLOSE 2
```

Végül elbúcsúzik és leáll:

```
910 PRINT:PRINT" *KESZ*"
999 END
```

Mentsük ki a programot SEQSZUK néven, és nincs akadálya annak, hogy kipróbáljuk:

– Listázzuk ki a KOLTSEGVISELOK állományunkat.

– Indítsuk el a SEQSZUK programot, és töröljünk 2-3 rekordot.

– Töröljük a KOLTSEGVISELOK állományt.

```
KOLTSEGVISELOK LISTAJA

KOD:    |  MEGNEVEZES:
--------+---------------------------------------------

KTSG --- KOLTSEGHELYI ALTALANOS

FELM --- HELYZETFELMERES

RTER --- RENDSZERTERVEZES

PTER --- PROGRAMTERVEZES

PKOD --- PROGRAMKODOLAS

TEST --- TESZTELES

ATAD --- RENDSZER ATADASA

MODO --- MODOSITAS

DOKU --- DOKUMENTALAS

GARA --- GARANCIALIS JAVITAS

--------+---------------------------------------------
```

```
KOLTSEGVISELOK ALLOMANY SZUKITESE

TORLENDO =? PKOD  TOROLHETO =I/N=? I
TORLENDO =? ATAD  TOROLHETO =I/N=? I
TORLENDO =? DOKU  TOROLHETO =I/N=? I
TORLENDO =? GARA  TOROLHETO =I/N=? I
*KESZ*
```

– Nevezzük át a SZUKITETT állományt KOLTSEGVISELOK-re.
– Nyomtassuk ki a KOLTSEGVISELOK állományt.

```
KOLTSEGVISELOK LISTAJA

KOD:    |  MEGNEVEZES:
--------+---------------------------------------------

KTSG --- KOLTSEGHELYI ALTALANOS
```

```
FELM --- HELYZETFELMERES

RTER --- RENDSZERTERVEZES

PTER --- PROGRAMTERVEZES

TEST --- TESZTELES

MODO --- MODOSITAS
```

MÓDOSÍTÁS*

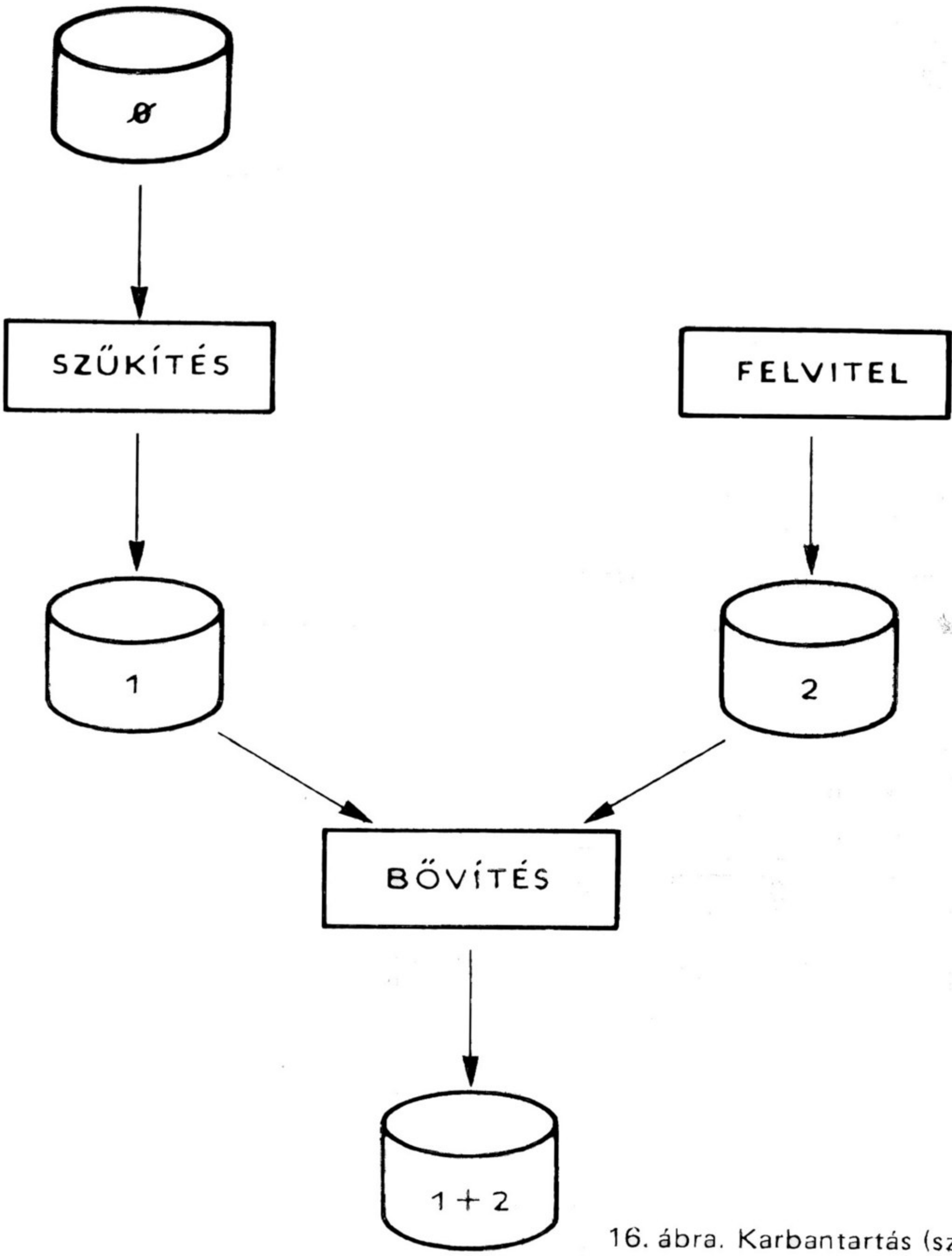
Költségviselők nemcsak keletkezhetnek és megszűnhetnek, hanem az adataik is változhatnak – nem beszélve arról, hogy az állományba hibás adatok is kerülhetnek. A módosítás tehát elkerülhetetlen.
Ha jól meggondoljuk, az eddig megismert adatfeldolgozási funkciókkal a KOLTSEGVISELOK állományt tetszés szerint módosíthatjuk (16. ábra).

Tudunk ugyanis rekordot beszúrni (bővítéssel) és rekordot törölni (szűkítéssel). Ezekre a műveletekre minden módosítás visszavezethető. Ha egy rekordban egy adat megváltozik, töröljük a teljes rekordot, majd egy újat viszünk fel helyette, persze újra megadva a rekord összes, tehát a megváltozott és a változatlanul maradt adatát.
Anélkül, hogy kipróbálnánk, beláthatjuk, hogy ez a módszer megelehetősen nehézkes és gazdaságtalan. De legfőképpen nem biztonságos. Amikor a hibás rekord helyett ismét megadjuk az új adatokat, újra követhetünk el hibákat.
Az interaktivitás azonban egy kényelmes és kevésbé veszélyes módosítási lehetőséget ad a kezünkbe.
Induljunk ki a SEQSZUK szűkítő programból. Ha ott a törölhető kérdésre nemleges választ adunk, lehetőség nyílik a módosításra. Csak annyit kell tennünk, hogy felülírjuk a rekord tartalmát, mielőtt a program felvinné az új állományba.
Ehhez felhasználhatjuk a SEQFELV felvivő program megfelelő moduljait.
Vegyük először a SEQSZUK szűkítő programot, azaz töltsük be (LOAD) és vezessük át rajta a következő módosításokat:

```
120 PRINT" KOLTSEGVISELOK ALLOMANY MODOSITASA"
220 OPEN 3,8,3,"MODOSITOTT,SEQ,WRITE"
320 INPUT" REKORD    = ";T$
460 IF V$="N" THEN GOSUB 1110:GOTO 510
```

Az utolsó sor a lényeges, mert az határozza meg, hogy ha a kiválasztott rekord nem törlendő, akkor felvitel előtt módosítani kell.
Ezután a 1110-es sortól kezdve gépeljük be a programba a SEQFELV felvivő program

*Az ismertetésre kerülő módszert a soros állományoknál viszonylag ritkán használják, mert az alapelve a közvetlen elérésű állományok karbantartására jellemző.



16. ábra. Karbantartás (szűkítés, bővítés)

110-től 510-ig terjedő sorait. (Egyszerűen minden sorszám elé egy 1-est gépelünk – így lesz a 110-ből 1110, a 121-ből 1121, és így tovább, az 510-ből 1510.)

```
1110 PRINT "▯":PRINT
1121 :   PRINT " KOLTSEGVISELO KODJA";
1122 :   PRINT
1123 :   PRINT:PRINT:PRINT
1131 :   PRINT " KOLTSEGVISELO MEGNEVEZESE:"
1132 :   PRINT:PRINT:PRINT
1141 :   PRINT " TERVEZETT RAFORDITAS:"
1142 :   PRINT:PRINT:PRINT
1151 :   PRINT " TENYLEGES RAFORDITAS:"
1152 :   PRINT:PRINT:PRINT
1310 :   PRINT "▣"
1320 :   PRINT "▣▣▣";"▮▮▮";K$
1321 :   INPUT "▯";K$
```

```
1322 :    K$=LEFT$(K$,4)
1330 :    PRINT "■■■■";"■■■";M$
1331 :    INPUT "■";M$
1332 :    M$=LEFT$(M$,30)
1340 :    PRINT "■■■■";"■■";T
1341 :    INPUT "■";T
1342 :    IF T<0    THEN T=0
1343 :    IF T>9999 THEN T=9999
1350 :    PRINT "■■■■";"■■";R
1351 :    INPUT "■";R
1352 :    IF R<0    THEN R=0
1353 :    IF R>9999 THEN R=9999
1410 :    PRINT "■"
1420 :    PRINT " EZ FOG A LEMEZRE KERULNI:"
1421 :    PRINT " -------------------------"
1422 :    PRINT
1430 :    PRINT
1431 :    PRINT:PRINT " ■KOOD■ = ";K$
1432 :    PRINT:PRINT " ■MEGN■ = ";M$
1433 :    PRINT:PRINT " ■TERV■ =" ;T
1434 :    PRINT:PRINT " ■TENY■ =" ;R
1435 :    PRINT
1440 :    INPUT " ■FELVIHETO■ =I/N=";W$
1441 :    IF W$="I" THEN GOTO 1510
1499 GOTO 1110
1510 PRINT#3,K$;S$;M$;S$;T;S$;R
1999 RETURN
```

Ezek a sorok hajtják végre az adatok bekérését és felvitelét. Némi módosításra azért itt is szükség volt. Ezekre kiemelten felhívjuk a figyelmet:

```
1122 :    PRINT
1440 :    INPUT" ■FELVIHETO■ =I/N=";W$
1441 :    IF W$="I" THEN GOTO 1510
1499 GOTO 1110
1510 PRINT#3,K$;S$;M$;S$;T;S$;R
1999 RETURN
```

Ezenkívül az 1323-as sort még *törölnünk* kell, hiszen itt nincs szükségünk a felírandó adatok végének figyelésére.

Megjegyezzük, hogy a VS válasz helyett azért használtunk W$-t, mert az előbbit a törlés/módosítás eldöntésére már lefoglaltuk, és nem célszerű, hogy a különböző jelzőink keveredjenek.

FIGYELEM!

Nagyon fontos az eljárás végén a RETURN utasítás. Az eljárást ugyanis szubrutinként GOSUB-bal hívjuk, és a hívás utáni visszatérést éppen a RETURN bitzosítja.

Az egész program együtt így néz ki:

```
100 REM:
101 REM: ===== ALLOMANY SZUKITES+MODOSITAS =====
102 REM:
110 PRINT"♫":PRINT
120 PRINT" KOLTSEGVISELOK ALLOMANY MODOSITASA"
130 PRINT:PRINT
140 S$=CHR$(13)
200 REM:
201 REM: ----- ELOKESZITES -----
202 REM:
210 OPEN 2,8,2,"KOLTSEGVISELOK,SEQ,READ"
220 OPEN 3,8,3,"MODOSITOTT,SEQ,WRITE"
300 REM:
301 REM: ----- MODOSITANDO BEKERESE -----
302 REM:
310 PRINT
320 :   INPUT" REKORD    =";T$
400 :   REM:
401 :   REM: ----- REKORD MEGKERESESE -----
402 :   REM:
410 :   INPUT#2,K$,M$,T,R
420 :   :   V=STATUS:V$=""
430 :   :   IF K$<>T$ THEN PRINT#3,K$;S$;M$;S$;T;S$;R:GOTO 510
434 :   :   REM:
435 :   :   REM: ----- TORLES -----
436 :   :   REM:
440 :   :   PRINT"♫";TAB(18);
450 :   :   :   INPUT" TOROLHETO =I/N=";V$
460 :   :   :   IF V$="N" THEN GOSUB 1110:GOTO 510
470 :   :   :   IF V$="I" THEN GOTO 510
480 :   :   GOTO 440
510 :   IF V=0 AND V$= "" THEN GOTO 410
520 IF V=0 AND V$<>"" THEN GOTO 310
600 REM:
601 REM: ----- BEFEJEZES -----
602 REM:
610 CLOSE 3
620 CLOSE 2
910 PRINT:PRINT" *KESZ*"
999 END


1000 REM:
1001 REM: ===== MODOSITAS VEGREHAJTASA =====
1002 REM:
1110 PRINT "♫":PRINT
1121 :   PRINT " KOLTSEGVISELO KODJA"
1122 :   PRINT
1123 :   PRINT:PRINT:PRINT
```

```
1131  :    PRINT " KOLTSEGVISELO MEGNEVEZESE:"
1132  :    PRINT:PRINT:PRINT
1141  :    PRINT " TERVEZETT RAFORDITAS:"
1142  :    PRINT:PRINT:PRINT
1151  :    PRINT " TENYLEGES RAFORDITAS:"
1152  :    PRINT:PRINT:PRINT
1310  :    PRINT "■"
1320  :    PRINT "■■■";"■■■";K$
1321  :    INPUT "■";K$
1322  :    K$=LEFT$(K$,4)
1330  :    PRINT "■■■";"■■■";M$
1331  :    INPUT "■";M$
1332  :    M$=LEFT$(M$,30)
1340  :    PRINT "■■■";"■■";T
1341  :    INPUT "■";T
1342  :    IF T<0    THEN T=0
1343  :    IF T>9999 THEN T=9999
1350  :    PRINT "■■■";"■■";R
1351  :    INPUT "■";R
1352  :    IF R<0    THEN R=0
1353  :    IF R>9999 THEN R=9999
1410  :    PRINT "■"
1420  :    PRINT " EZ FOG A LEMEZRE KERULNI:"
1421  :    PRINT " ----------------------------------"
1422  :    PRINT
1430  :    PRINT
1431  :    PRINT:PRINT " ■KOOD■ = ";K$
1432  :    PRINT:PRINT " ■MEGN■ = ";M$
1433  :    PRINT:PRINT " ■TERV■ =" ;T
1434  :    PRINT:PRINT " ■TENY■ =" ;R
1435  :    PRINT
1440  :    INPUT " ■FELVIHETO■ =I/N=";W$
1441  :    IF W$="I" THEN GOTO 1510
1499 GOTO 1110
1510 PRINT#3,K$;S$;M$;S$;T;S$;R
1999 RETURN
```

Mentsük ki SEQMODO néven, majd próbáljuk ki.

Ha megadunk egy költségviselőt, és a törölhető kérdésre nemleges választ adunk, tetszés szerint módosíthatjuk az adatait, akárhányszor egymás után. Ha valamely adatot nem kívánunk módosítani, csak a RETURN gombot kell megnyomnunk.

```
KOLTSEGVISELOK ALLOMANY MODOSITASA

REKORD    =? FELM  TOROLHETO =I/N=? I
REKORD    =? PTER  TOROLHETO =I/N=? N
```

```
EZ FOG A LEMEZRE KERULNI:
------------------------------------


[illegible] = PTER
[illegible] = PROGRAMTERVEZES
[illegible] = 600
[illegible] = 800
[illegible] =I/N=? I
```

A rekord akkor kerül fel a MODOSITOTT állományba, ha a *felvihető* kérdésre igenlő választ adtunk.
A program érdekessége, hogy a módosítás, akárcsak a felvitelnél, rendkívül kényelmes, és főleg biztonságos. Ha a *felvihető* kérdésre nemleges választ adunk, addig módosíthatjuk a rekordot, amíg a tartalmával tökéletesen meg nem vagyunk elégedve.

FIGYELEM!
Ha tévedésből adtunk nemleges választ a törlendő kérdésre, nem feltétlenül kell a rekordot módosítanunk, adhatunk minden adatra közvetlenül RETURN-t. Így az új állományba a rekord változatlanul kerül fel. De mindenképpen felkerül – az ilyen rekordot törölni már nem lehet, illetve lehet, de csak a következő programfutás alkalmával.

Megjegyezzük, hogy ez a módosító program magában foglalja a szűkítést is, azaz a rekordok törlésére is alkalmas – csupán a törölhető kérdésre kell igenlő választ adnunk.
A program végrehajtása után törölhetjük a KOLTSEGVISELOK állományt, és átnevezhetjük a MODOSITOTT állományt KOLTSEGVISELOK-re.
Óva intünk mindenkit attól, hogy ezt a programot végső megoldásnak tekintse. Sokkal inkább kiindulópont, amely az interaktív módosításnak csak az alapjait mutatja be, nem pedig az összes lehetőségét. (Ezekre egyébként a random és relatív állományokkal foglalkozó kötetünkben bővebben kitérünk.)
A finomítást kezdhetjük például azzal, hogy töröljük a képernyőt a módosítás után. Ez csak ennyi:

```
1520 PRINT"[CLR]":PRINT
```

A többit az olvasóra bízzuk.
Annyit még hozzáteszünk, hogy a rekord adatonkénti módosítása akkor egyszerű, ha a rekord adatai jól áttekinthetően elférnek a képernyőn – amint ez a példánkban is látható volt. Ellenkező esetben a rekordot több részletben kell a képernyőre kihoznunk, és meg kell szerveznünk a lapozást, azaz a képernyőváltást.

A programba a beszúrás is beépíthető. Ehhez az kell, hogy a törölhető kérdésre adott tagadó válasz után a módosító szubrutin megkérdezze, hogy beszúrást vagy módosítást akarunk-e végrehajtani. (Az utóbbi esetben a már ismert módon jár el.) Beszúrás esetén felviszi az aktuális rekordot, majd egy üres rekord módosítását hajtja végre. Ennek rendezett állományoknál van jelentősége, mert így a beszúrt rekordok rendezés nélkül a helyükre kerülhetnek.

FELÜLÍRÁS

Elég kellemetlen, hogy bizonyos adatfeldolgozó műveletek során újabb és újabb állományok keletkeznek, így ezek törlésére és átnevezésére állandóan figyelmet kell fordítanunk.

A probléma egy részét megoldaná, ha bizonyos meglevő állományokat felülírhatnánk. Ez elvileg lehetséges:

```
111 OPEN 2,8,2,"@:MODOSITO,SEQ,WRITE"
```

Itt a megnyitás a szokásos, de szerepel benne a SAVE parancsból már ismert felülírás (kukac = @) jel. Hatására az állomány felülíródik az új állománnyal.

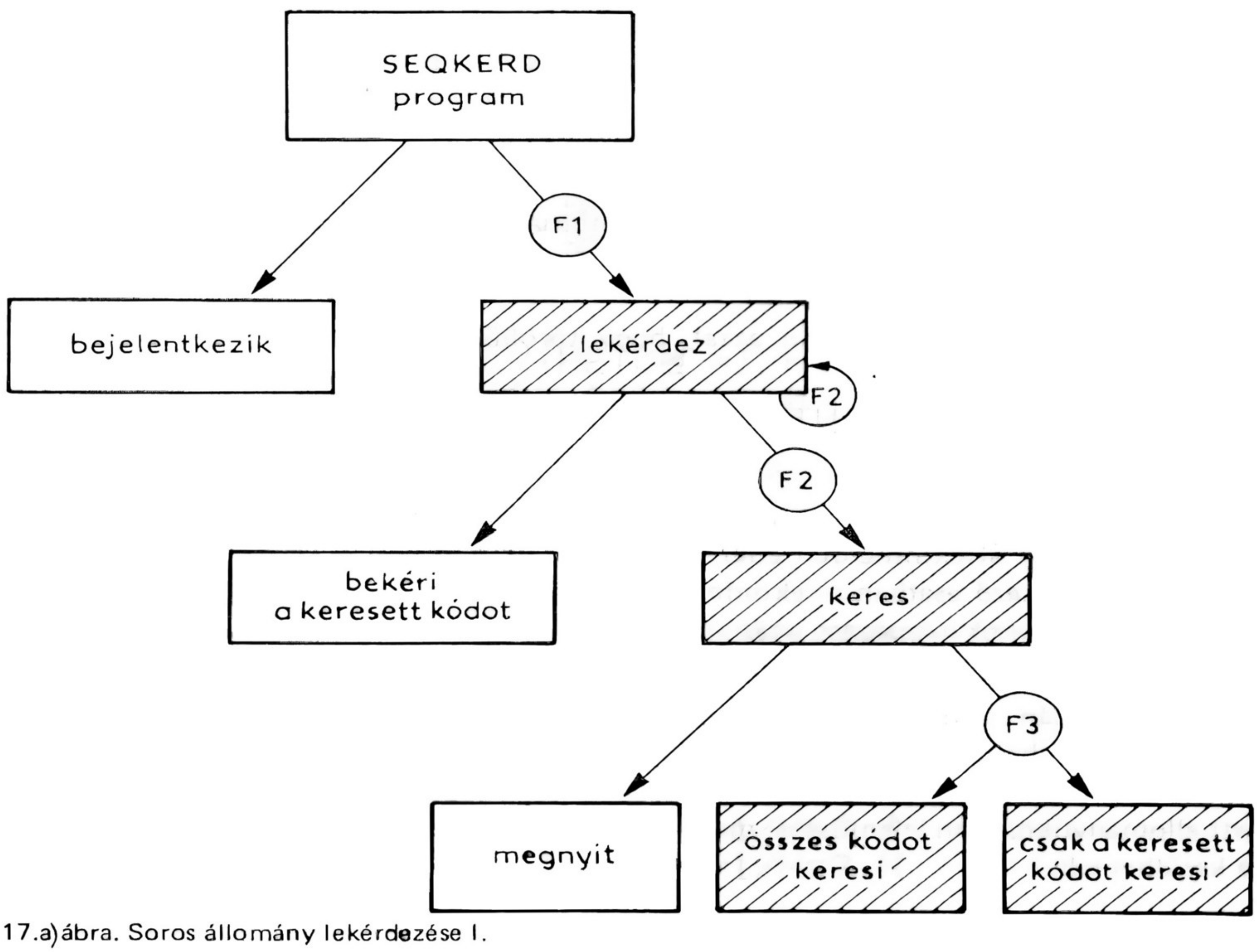
FIGYELEM!

Semmilyen hibaüzenetet nem kapunk, ha a felülírandó állomány nem létezik. Ilyenkor az állomány ugyanúgy jön létre, mintha írásra nyitottuk volna meg. Ezért megtehettük volna, hogy a SEQFELV felvivő programba ilyen megnyitást kódolunk. Nem tettük, mert a felülírás veszélyes művelet. Sikertelensége esetén a már meglevő állományunk is tönkremehet.

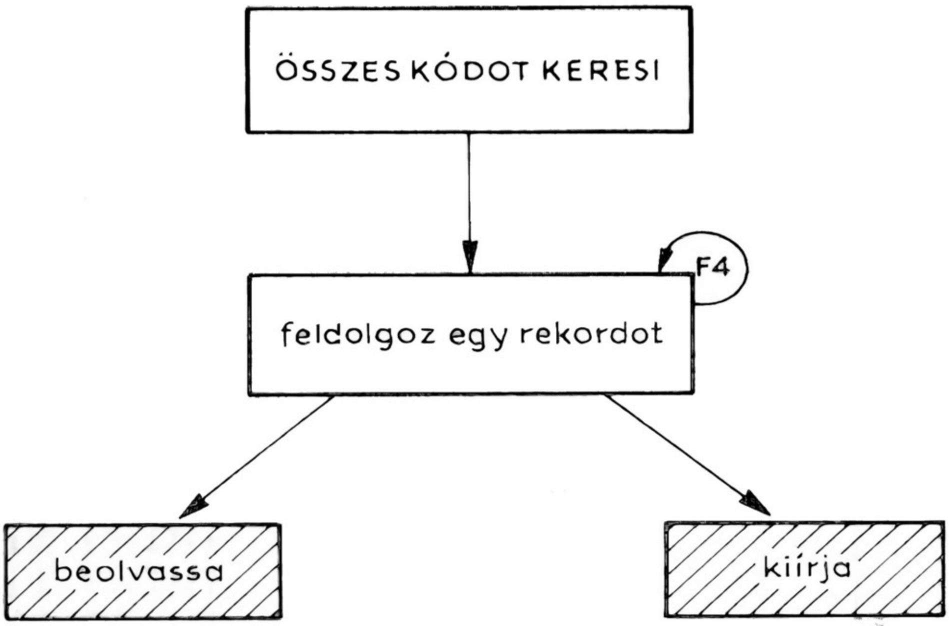
LEKÉRDEZÉS

A soros állományokra persze az egyszerű kinyomtatásnál bonyolultabb feldolgozások is végrehajthatók. Tekintsük például a soros állomány lekérdezését.

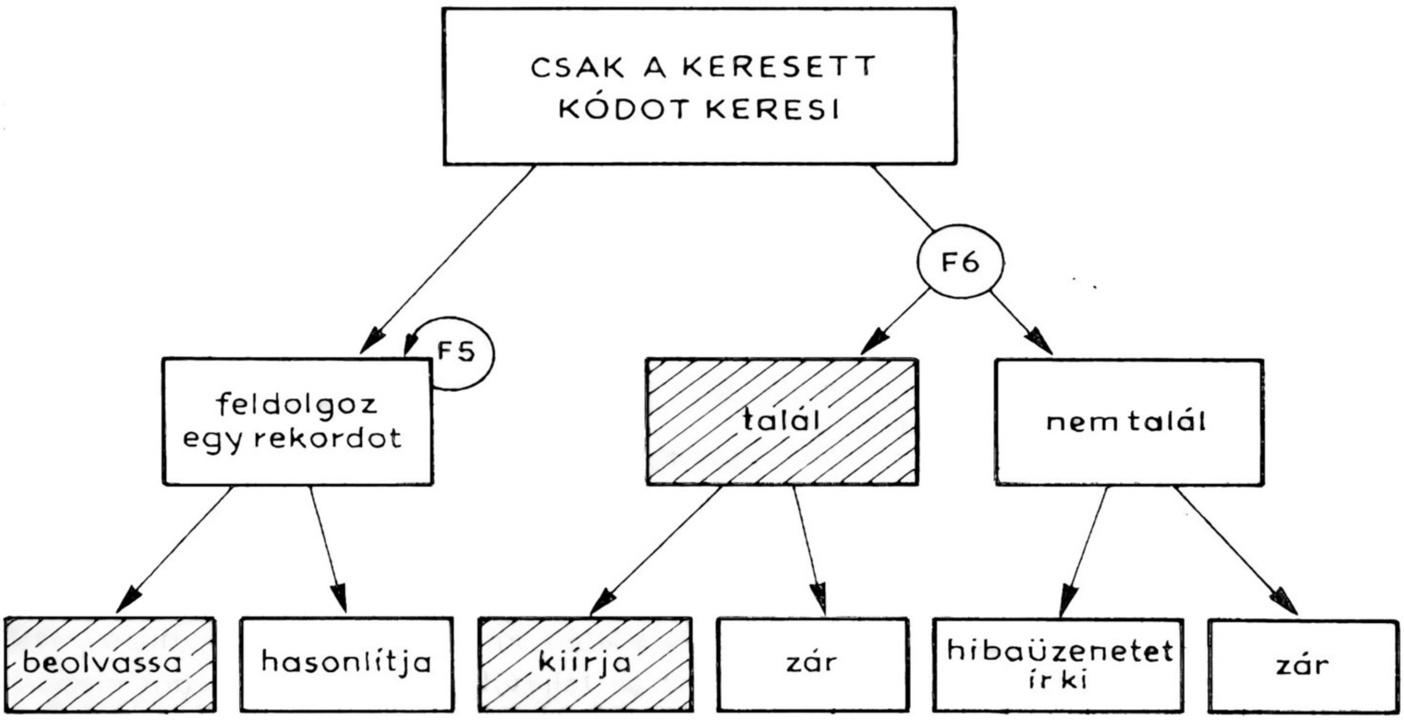
A soros lekérdezés lényege, hogy a soros állomány rekordjait csak olvassuk; egyetlen adatot sem változtatunk meg. Az állományból kivett adatokat nem dolgozzuk fel, azokra műveletet nem hajtunk végre, csak megjelenítjük – képernyőn vagy nyomtatón.



17.a) ábra. Soros állomány lekérdezése I.



17. b) ábra. Soros állomány lekérdezése II.



17. c) ábra. Soros állomány lekérdezése III.

A feltételek:

F1: ha mehet = "I"

F2: ha az X$ keresett kód nem = "––––"

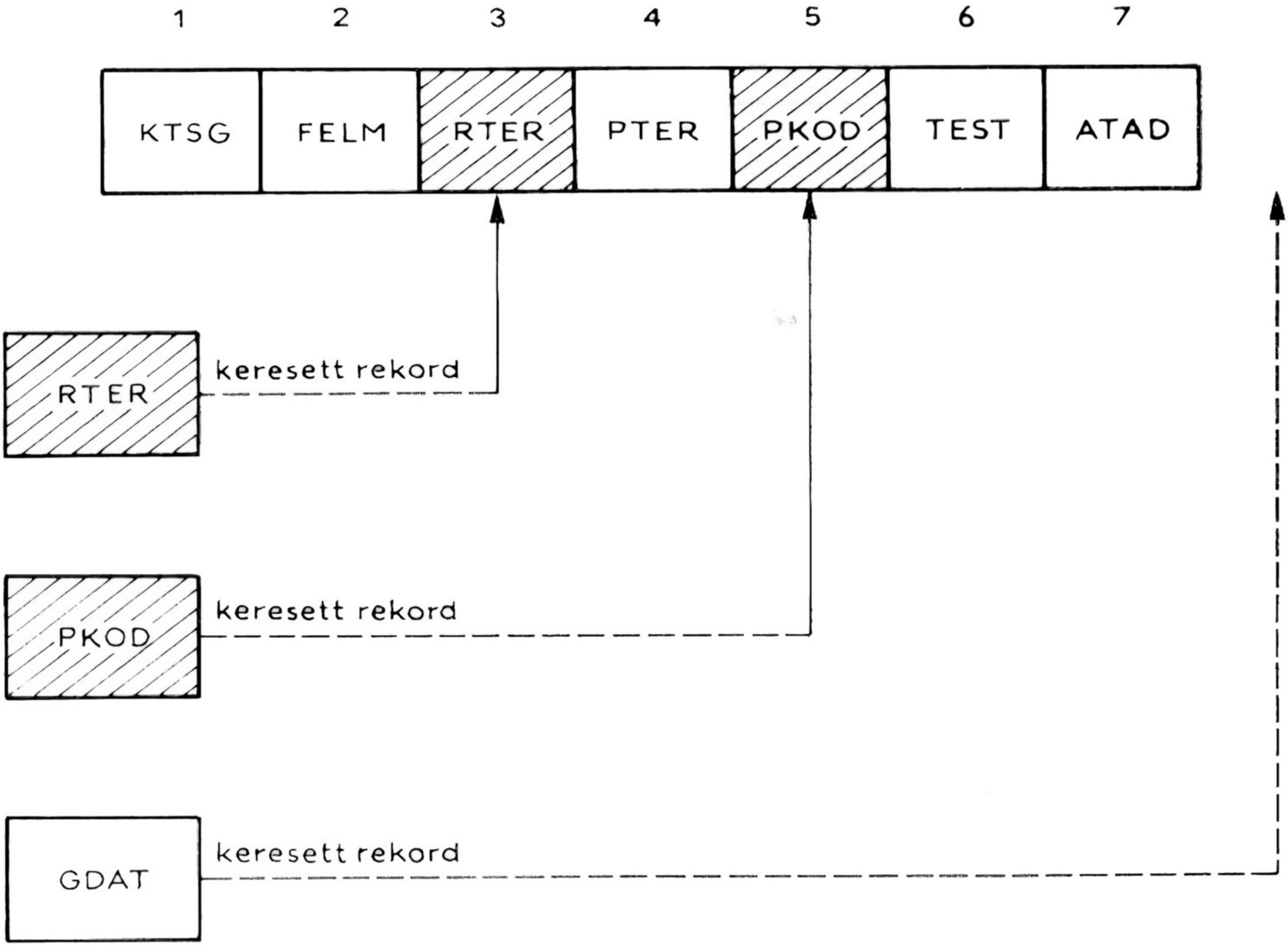
F3: ha az X$ keresett kód = "~~xxxx~~"

F4: ha van még adat, azaz a K$ kód nem üres

F5: ha van még adat az állományban, azaz a K$ kód nem üres, és a beolvasott K$ kód nem azonos a keresett X$ kóddal

F6: ha a beolvasott K$ kód azonos a keresett X$ kóddal

A lekérdezés sajátossága, hogy általában nem az összes adatra vagyunk kíváncsiak; az állományból csak azokat az adatokat (rekordokat) jelenítjük meg, amelyek megfelelnek bizonyos kritériumoknak, azaz kielégítenek valamilyen lekérdezési szempontot vagy szempontokat. (Ilyenkor tulajdonképpen már válogatást hajtunk végre.)
A lekérdezés fontos eleme tehát a keresés, amelynek célja az adott lekérdezési szempontot kielégítő adat vagy adatok megtalálása.



18. ábra. Soros keresés

A soros keresés sajátossága, hogy az állomány adatait sorban olvassuk, és minden olvasás után megvizsgáljuk a lekérdezési feltételt. Ha az teljesült, a keresett rekordot megtaláltuk. Ha nem, tovább kell keresnünk.
A keresést minden egyes lekérdezésre külön végre kell hajtanunk, és mindig az állomány elején kell kezdenünk, és addig kell folytatnunk, amíg meg nem találjuk a keresett rekordot, vagy amíg el nem érjük az állomány végét. Csak az utóbbi esetben lehetünk biztosak abban, hogy a keresett rekord nincs az állományunkban.
NEW parancs kiadása után gépeljük be a következő lekérdező programot. Ez kétféle lekérdezést hajt végre: egyedit és általánost. Azaz akárhányszor ki tudja keresni az állomány bármelyik rekordját, tetszőleges sorrendben; illetve egyenként meg tudja jeleníteni az állomány összes rekordját, előfordulásuk sorrendjében. A kétféle funkció közül a felhasználó tetszése szerint választhat, de a funkciókat nem keverheti.
A program először is bejelentkezik:

```
1 PRINT "□"
2 PRINT
3 PRINT " KOLTSEGVISELOK LEKERDEZESE"
4 PRINT " ----------------------------"
5 PRINT
6 PRINT
7 INPUT " MEHET =I/N=";V$
8 IF V$="N" THEN GOTO 999
9 IF V$="I" THEN GOTO 110
10 GOTO 6
```

Majd kéri a keresendő rekord X$ kulcsát, azaz a költségviselő kódját:

```
110 PRINT "□"
111 PRINT
120 PRINT " ADJA MEG ANNAK A"
121 PRINT
130 PRINT " KOLTSEGVISELONEK A KODJAT,"
131 PRINT
140 PRINT " AMELYIKNEK AZ ADATAIT KERI"
141 PRINT
142 PRINT
150 PRINT " (KOD=****, HA MINDEGYIKET)"
151 PRINT
160 PRINT " (KOD=----, HA EGYIKET SEM)"
161 PRINT
162 PRINT
163 PRINT
170 INPUT " ■KOLTSEGVISELO■ = ";X$
```

Ha a keresett X$ kód négy kötőjel, a program leáll:

```
210 :   IF X$="----" THEN GOTO 910
```

Ha nem, akkor olvasásra megnyitja a KOLTSEGVISELOK állományt:

```
220 :   OPEN 2,8,2,"KOLTSEGVISELOK,SEQ,READ"
```

Ha a keresett X$ kód se nem négy csillag, se nem négy kötőjel, akkor keresni kezdi az állományban, vagyis olvasni kezdi a rekordokat:

```
310 :   K$=""
320 :   :   INPUT#2,K$,M$,T,R
```

Megjegyezzük, hogy a K$ kódot azért állítjuk be üres karakterre, mert az olvasás sikerességéről így győződhetünk meg a legegyszerűbben.

Ha ugyanis az olvasás után a költségviselő K$ kódja üres marad, akkor az olvasás sikertelen volt. Feltételezzük, hogy ezt nem hiba okozta, hanem amiatt következett be, mert elértük az állomány végét. Ilyenkor tehát a keresett rekord nincs az állományban:

```
330 :   :   IF K$="" THEN GOTO 360
```

Ha a K$ kód nem üres, akkor az olvasás sikeres volt. Ilyenkor a program megnézi, hogy a beolvasott K$ kód megegyezik-e a keresett X$ kóddal. Ha igen, a rekordot megtaláltuk:

```
340 :   :   IF K$=X$ THEN GOTO 410
```

Ha nem, akkor tovább kell keresni:

```
350 :   GOTO 310
```

Ha a keresett rekord nincs az állományban, akkor a program ilyen értelmű M$ üzenetet állít be:

```
360 :   K$=X$
370 :   M$="NINCS NYILVANTARTVA"
380 :   T=0
390 :   R=0
```

Majd lezárja az állományt, és megjeleníti a képernyőn a rekodot, ami most az imént beállított üzenetet tartalmazza:

```
410 :   CLOSE 2
420 :   GOSUB 1010
```

Ugyanezt teszi, ha a rekordot megtalálta, de ilyenkor nem állít be előzetesen üzenetet, így a megjelenített rekord a lemezen levő K$, M$, T, R adatokat tartalmazza.
Ezután mindkét esetben újra kezdi a lekérdezést:

```
499 GOTO 110
```

Ha az összes rekord kiíratását választottuk, azaz a keresett X$ kód négy csillag volt, akkor a program ugyanígy keres:

```
230 :   IF X$="****" THEN GOTO 510
510 :   K$=""
520 :   :   INPUT#2,K$,M$,T,R
530 :   :   IF K$="" THEN GOTO 610
```

Nem vizsgálja azonban, hogy a beolvasott K$ kód azonos-e a keresettel, hanem mindenképpen megjeleníti a rekordot, majd veszi a következőt:

```
540 :   :   GOSUB 1010
599 :   GOTO 510
```

Ha a keresett X$ kód leállítójel, azaz négy kötőjel volt, a program kijelentkezik:

```
910 PRINT
920 PRINT " A LEKERDEZESNEK VEGE"
```

Majd leáll:

```
999 END
```

A beolvasott rekord adatait külön alprogram, szubrutin jeleníti meg a képernyőn, áttekinthető formában:

```
1010 PRINT "▯"
1020 PRINT
1030 PRINT " KOLTSEGVISELO = ";X$
1040 PRINT " ------------------------------------------------------------"
1050 PRINT
1060 PRINT
1070 PRINT " KOOD = ";K$
1080 PRINT
```

```
1090 PRINT " MEGN = ";M$
1100 PRINT
1110 PRINT " TERV =" ;T
1120 PRINT
1130 PRINT " TENY =" ;R
1140 PRINT
1150 PRINT
1160 PRINT " ------------------------------------------------"
```

Az adatokat tetszés szerinti ideig tanulmányozhatjuk, akár egyedi rekordot kérdezünk le, akár sorban az összeset; a program ugyanis türelmesen vár, amíg a RETURN billentyűt meg nem nyomjuk:

```
1170 PRINT " NYOMJA MEG A ■RETURN■";
1180 PRINT " GOMBOT, HA"
1190 PRINT
1200 INPUT " JOHET A KOVETKEZO";V$
1999 RETURN
```

Ha az összes rekordot kértük le, így tetszés szerint lapozhatunk az állomány rekordjai között, de mindig csak egyesével, és mindig csak előre.

Ha össze-vissza akarunk lapozni, akkor az egyedi lekérdezést kell választanunk.

Megjegyezzük, hogy bármilyen gombot nyomunk meg a RETURN előtt, azt a program figyelmen kívül hagyja. Egyébként azért választottuk a RETURN billentyűt, mert annak a megnyomását tartottuk a legkézenfekvőbbnek. (Használhattuk volna az INPUT helyett a GET utasítást is, de az ilyen programozástechnikai kérdések nem a lemezkezelés témakörébe tartoznak.)

Íme a teljes program:

```
1 PRINT "♥"
2 PRINT
3 PRINT " KOLTSEGVISELOK LEKERDEZESE"
4 PRINT " ---------------------------"
5 PRINT
6 PRINT
7 INPUT " MEHET =I/N=";V$
8 IF V$="N" THEN GOTO 999
9 IF V$="I" THEN GOTO 110
10 GOTO 6
100 REM:
101 REM: ------------------------------------------- KERESETT KOD BEKERESE
102 REM:
110 PRINT "♥"
111 PRINT
120 PRINT " ADJA MEG ANNAK A"
121 PRINT
130 PRINT " KOLTSEGVISELONEK A KODJAT,"
131 PRINT
140 PRINT " AMELYIKNEK AZ ADATAIT KERI"
141 PRINT
142 PRINT
150 PRINT " (KOD=****, HA MINDEGYIKET)"
151 PRINT
160 PRINT " (KOD=----, HA EGYIKET SEM)"
161 PRINT
```

```
162 PRINT
163 PRINT
170 INPUT " ▪KOLTSEGVISELO■ = ";X$
200 :   REM:
201 :   REM: ---------------------------------------- ELOKESZITES
202 :   REM:
210 :   IF X$="----" THEN GOTO 910
220 :   OPEN 2,8,2,"KOLTSEGVISELOK,SEQ,READ"
230 :   IF X$="****" THEN GOTO 510
300 :   REM:
301 :   REM: ---------------------------------------- REKORDKERESES
302 :   REM:
310 :   K$=""
320 :   :   INPUT#2,K$,M$,T,R
330 :   :   IF K$="" THEN GOTO 360
340 :   :   IF K$=X$ THEN GOTO 410
350 :   GOTO 310
354 :   REM:
355 :   REM: ---------------------------------------- HA NEM TALAL
356 :   REM:
360 :   K$=X$
370 :   M$="NINCS NYILVANTARTVA"
380 :   T=0
390 :   R=0
400 :   REM:
401 :   REM: ---------------------------------------- HA TALAL
402 :   REM:
410 :   CLOSE 2
420 :   GOSUB 1010
499 GOTO 110
500 :   REM:
501 :   REM: ---------------------------------------- TELJES LEKERDE
502 :   REM:
510 :   K$=""
520 :   :   INPUT#2,K$,M$,T,R
530 :   :   IF K$="" THEN GOTO 610
540 :   :   GOSUB 1010
599 :   GOTO 510
600 REM:
601 REM: ------------------------------------------ ATTERES UJ REKORDR
602 REM:
610 CLOSE 2
620 GOTO 110
900 REM:
901 REM: ------------------------------------------ BEFEJEZES
902 REM:
910 PRINT
920 PRINT " A LEKERDEZESNEK VEGE"
999 END
1000 REM:
1001 REM: ======================================== REKORD KIIRASA
1002 REM:
1010 PRINT "▯"
1020 PRINT
1030 PRINT " KOLTSEGVISELO = ";X$
1040 PRINT " ---------------------------------------"
1050 PRINT
1060 PRINT
1070 PRINT " KOOD = ";K$
1080 PRINT
1090 PRINT " MEGN = ";M$
1100 PRINT
```

```
1110 PRINT " TERV =" ;T
1120 PRINT
1130 PRINT " TENY =" ;R
1140 PRINT
1150 PRINT
1160 PRINT " ------------------------------------------------------------------------"
1170 PRINT " NYOMJA MEG A RETURN";
1180 PRINT " GOMBOT, HA"
1190 PRINT
1200 INPUT " JOHET A KOVETKEZO";V$
1999 RETURN
```

Mentsük ki most ezt a programot is SEQKERD néven, a SOROS lemezünkre! Ezután indítsuk el!

Először kérjünk teljes körű lekérdezést, azaz adjunk meg négy csillagot keresett kódként.

Lapozzuk végig az állományt, majd amikor a végére értünk, hajtsunk végre egyedi lekérdezéseket az első, az utolsó és valamelyik közbülső kódra.

Ezután keresett kódként adjuk meg a négy kötőjelet. Ekkor a program leáll.

```
ADJA MEG ANNAK A
KOLTSEGVISELONEK A KODJAT,
AMELYIKNEK AZ ADATAIT KERI

(KOD=****, HA MINDEGYIKET)
(KOD=----, HA EGYIKET SEM)

KOLTSEGVISELO = ?

KOLTSEGVISELO = ATAD
----------------------------------------

KOOD = ATAD
MEGN = RENDSZER ATADASA
TERV = 50
TENY = 50

----------------------------------------
NYOMJA MEG A RETURN GOMBOT, HA
JOHET A KOVETKEZO?

KOLTSEGVISELO = TTTT
----------------------------------------

KOOD = TTTT
MEGN = NINCS NYILVANTARTVA
```

```
TERV = 0
TENY = 0
--------------------------------------------------------------------
NYOMJA MEG A RETURN GOMBOT, HA
JOHET A KOVETKEZO?

KOLTSEGVISELO = PKOD
--------------------------------------------------------------------

KOOD = PKOD
MEGN = PROGRAMKODOLAS
TERV = 300
TENY = 200
--------------------------------------------------------------------
NYOMJA MEG A RETURN GOMBOT, HA
JOHET A KOVETKEZO?
```

Megjegyezzük, hogy ebben a programban elöltesztelő ciklust alkalmaztunk a kereséshez. Megtehettük, mert nem a STATUS jelzőt használtuk az állomány végének a figyeléséhez, hanem a lemezkezelő azon tulajdonságát használtuk ki, hogy az állomány vége után már nem olvas többet, így a feltöltendő változóban ilyenkor megmarad a korábbi érték.
E módszer a valamilyen hiba következtén sikertelenül végrehajtott olvasást ugyanúgy kezeli, mintha állomány vége lett volna, azaz mindig csak az állomány olvasható részében keresünk.

VÁLOGATÁS

A lekérdezés, főleg a nem teljes körű, hanem a különböző feltételekhez kötött lekérdezés, a tulajdonképpeni *válogatás*, igen fontos művelet.
Itt jöhetnek elő a számítógépes tárolás és feldolgozás igazi előnyei. Egy dolgozó személyi adatait rávezetni egy kartonra ugyanis nem lényegesen nagyobb munka, mint felvinni egy nyilvántartó állományba. De kiválogatni a kartonokról azokat a dolgozókat, akik 30 éven aluliak, nők, nem Budapesten laknak, 5 évnél régebben léptek be, és legalább 2 gyerekük van, manuálisan végrehajtva szinte megoldhatatlan feladat, és ha sikerül is, nem lehetünk biztosak benne, hogy az eredmény hibátlan. Feltéve pesze, hogy nem pár tucatnyi dolgozóról van szó.

Míg tehát egy adott dolgozó kartonját kiemelni a nyilvántartásból kézzel sokkal gyorsabban lehet, mint géppel, az összes karton végigválogatását a gép gyorsabban és megbízhatóbban végzi el.
Ugyanez a helyzet a költségviselőinkkel is. A gépi válogatás szinte ugyanannyi időt vesz igénybe, mint az állomány válogatás nélküli lekérdezése, hiszen mindkét esetben egyszer kell végigolvasni az állományt, és az olvasási idő mellett annak a néhány

IF utasításnak a végrehajtása, amely a válogatási szempontokat megvizsgálja, alig jelent idővesztességet.

Nézzünk egy példát! Módosítsuk a SEQERD programot az alábbi módon:

```
240 L=LEN(X$)
340 IF LEFT$(K$,L)=X$ THEN GOTO 410
370 M$="NINCS (TOBB) NYILVANTARTVA"
410 GOSUB 1010
420 IF LEFT$(M$,5)="NINCS" THEN GOTO 490
430 GOTO 310
490 CLOSE 2
```

Ezek a módosítások a programba így épülnek be:

```
200 :  REM:
201 :  REM: ------------------------------------------ ELOKESZITES
202 :  REM:
210 :  IF X$="----" THEN GOTO 910
220 :  OPEN 2,8,2,"KOLTSEGVISELOK,SEQ,READ"
230 :  IF X$="****" THEN GOTO 510
240 :  L=LEN(X$)
300 :  REM:
301 :  REM: ------------------------------------------ REKORDKERESES
302 :  REM:
310 :  K$=""
320 :  :  INPUT#2,K$,M$,T,R
330 :  :  IF K$="" THEN GOTO 360
340 :  :  IF LEFT$(K$,L)=X$ THEN GOTO 410
350 :  GOTO 310
354 :  :  REM:
355 :  :  REM: ------------------------------------- HA NEM TALAL
356 :  :  REM:
360 :  :  K$=X$
370 :  :  M$="NINCS (TOBB) NYILVANTARTVA"
380 :  :  T=0
390 :  :  R=0
400 :  :  REM:
401 :     REM: ------------------------------------- HA TALAL
402 :  :  REM:
410 :  :  GOSUB 1010
420 :  :  IF LEFT$(M$,5)="NINCS" THEN GOTO 490
430 :  GOTO 310
490 CLOSE 2
499 GOTO 110
```

Mentsük ki a programot SEQVALO néven, majd próbáljuk ki. Ha a keresett X$ kódként csak egy P betűt adunk meg, akkor a 340 IF a beolvasott K$ kódnak csak az első karakterét fogja vizsgálni. Az ennek megfelelő, vagyis a P-vel kezdődő azonosítójú rekordot a 410 GOSUB kiírja, majd a program a 430 GOTO hatására tovább keres. Mindaddig, amíg el nem éri az állomány végét, amikor is NINCS (TOBB) NYILVANTARTVA jelzést ad. Vagyis a program ki fogja válogatni az állományból az összes olyan rekordot, amelynek a kódja P betűvel kezdődik.

Hasonló módon átírhatnánk a programot úgy is, hogy ne a kód szerint válogasson, hanem valamely másik adat, mondjuk a tervezett ráfordítás szerint.

```
KOLTSEGVISELO = P
--------------------------------------------

KOOD = PTER
MEGN = PROGRAMTERVEZES
TERV = 600
TENY = 800

--------------------------------------------
NYOMJA MEG A RETURN GOMBOT, HA
JOHET A KOVETKEZO?

KOLTSEGVISELO = P
--------------------------------------------

KOOD = PKOD
MEGN = PROGRAMKODOLAS
TERV = 300
TENY = 200

--------------------------------------------
NYOMJA MEG A RETURN GOMBOT, HA
JOHET A KOVETKEZO?

KOLTSEGVISELO = P
--------------------------------------------

KOOD = P
MEGN = NINCS (TOBB) NYILVANTARTVA
TERV = 0
TENY = 0

--------------------------------------------
NYOMJA MEG A RETURN GOMBOT, HA
JOHET A KOVETKEZO?
```

Az sem okoz komoly nehézséget, hogy *ne egy, hanem több adat szerint válogassunk,* csupán nem egy, hanem több keresendő adatot kell bekérnünk, és ennek megfelelően nem egy, hanem több válogatási feltételt kell megadnunk. Az ilyen összetett feltételeket vagy több IF utasításból álló feltételrendszerrel, vagy egyetlen IF utasításban elhelyezett logikai kifejezéssel kódolhatjuk (vagy szükség esetén a kettő kombinációjával).

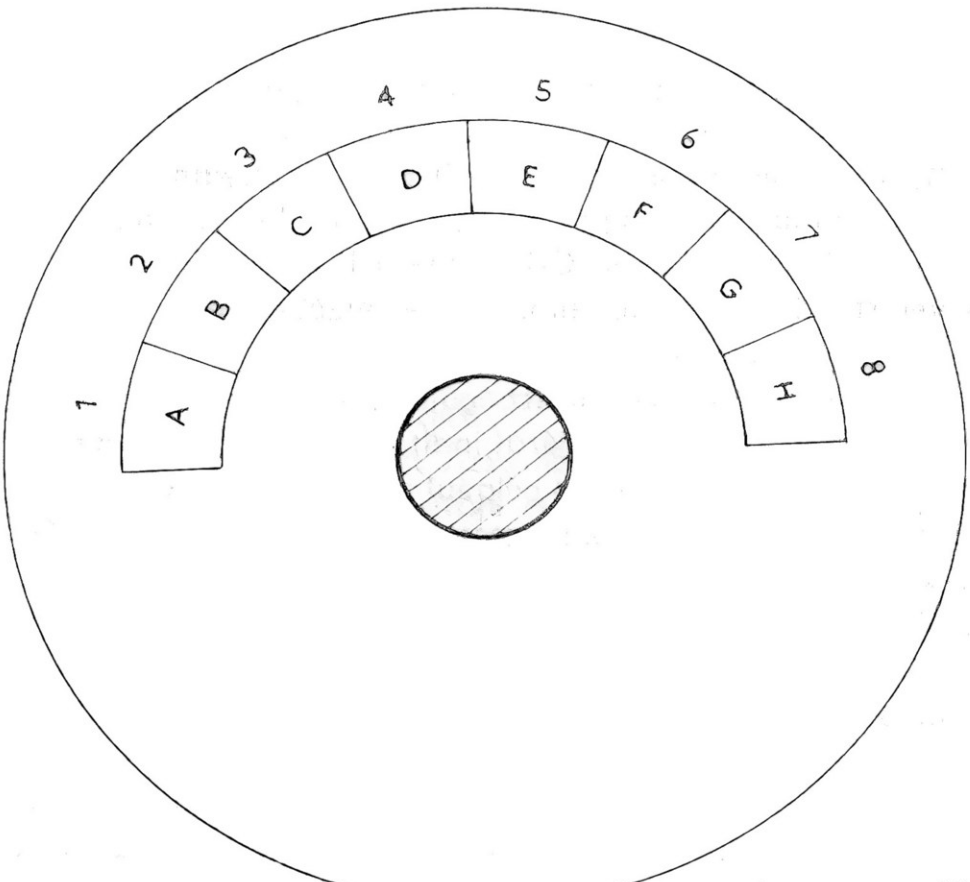
A relációkban természetesen nemcsak az egyenlőségjel szerepelhet. Például:

```
340 IF RIGHT$(M$,L)=X$ AND (R>T OR R>100) THEN GOTO 410
```

Hatására a program kiválogatja azokat a költségviselőket, amelyeknek a megnevezése egy adott karaktersorozatra, mondjuk „TANFOLYAM"-ra végződik, és amelyeknél a tényleges ráfordítás meghaladta a tervezettet, vagy a 100 órát, vagy mindkettőt.

Rendezett állományok

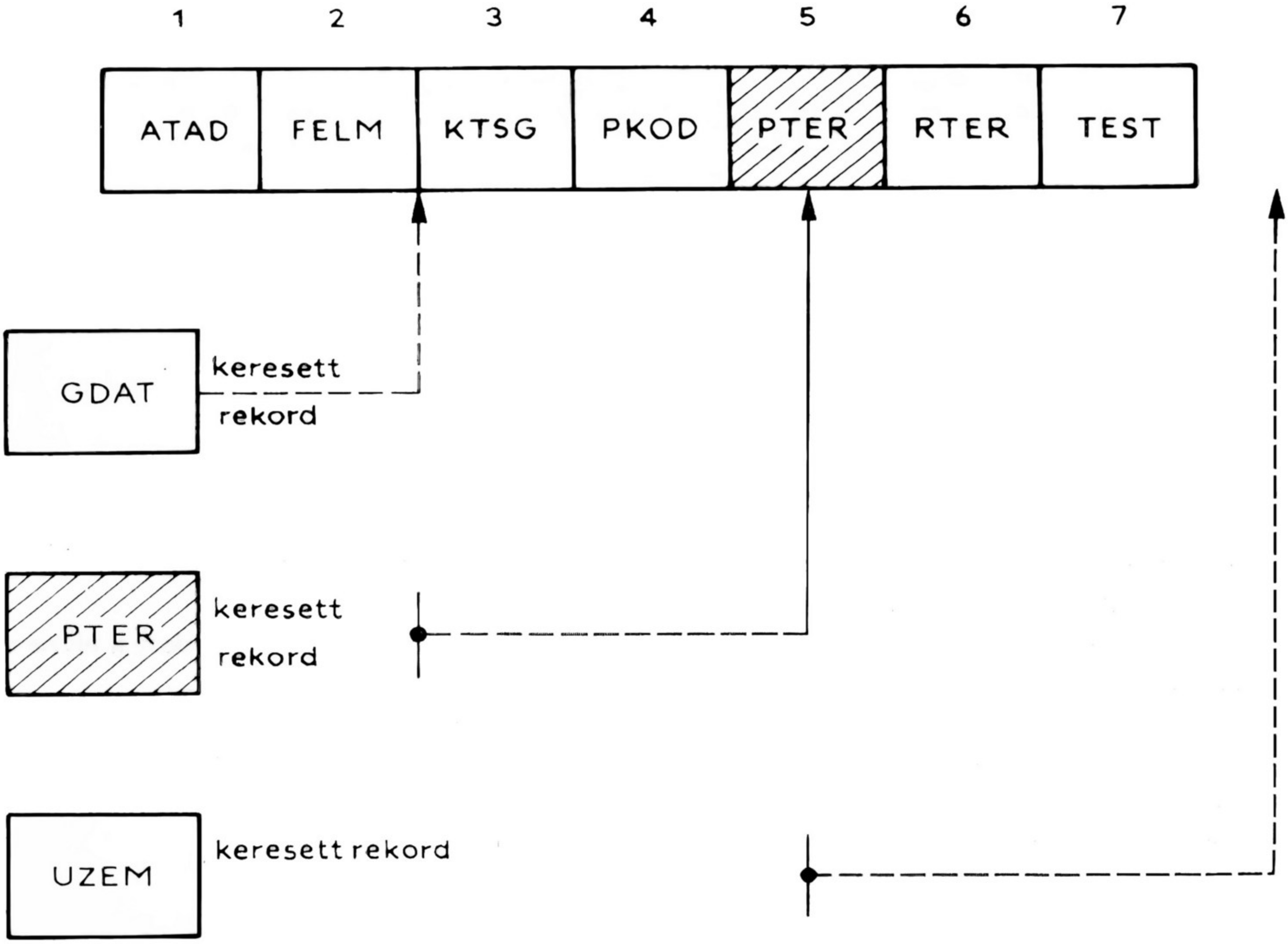
A rendezett állomány tulajdonképpen nem más, mint egy speciális soros állomány (19. ábra).
Ebben a rekordok fizikai sorrendje megegyezik az állomány jellegéből fakadó természetes, vagy az általunk a rekordoknak tulajdonított mesterséges logikai sorrenddel.
Azt az elvet, amely szerint a rekordok logikai sorrendjét meghatározzuk, rendezési elvnek nevezzük.
A példánkban a rekordok alfabetikusan növekvő sorrendben rendezettek.
Hasonló módon rendezhetjük a KOLTSEGVISELOK állományt is; a rekordazonosító, azaz a költségviselő kódja szerint.
Természetesen a rendezettség befolyásolja az állomány kezelését.



19. ábra. Rendezett állomány

KERESÉS

Vegyük például a keresést (20. ábra):



20. ábra. Keresés rendezett állományban

Az első szembeötlő sajátosság az, hogy már akkor meggyőződhetünk arról, hogy egy keresett rekord nincs az állományban, amikor megtaláljuk az első olyan rekordot, amelynek azonosítója nagyobb a keresettnél. (Ilyen például a GDAT rekord, amelyet a KTSG rekord után már nem érdemes keresnünk, hiszen az állomány e részében egyre csak nagyobb azonosítójú rekordokra fogunk bukkanni.)

Ha viszont a keresendő rekordok is ugyanolyan rendezettségűek, mint maga az állomány, akkor nem kell minden egyes keresést az állomány elejéről indítanunk, hanem folytathatjuk onnan, ahol az előző keresésnél abbahagytuk. (Lásd például a PTER rekordot!)

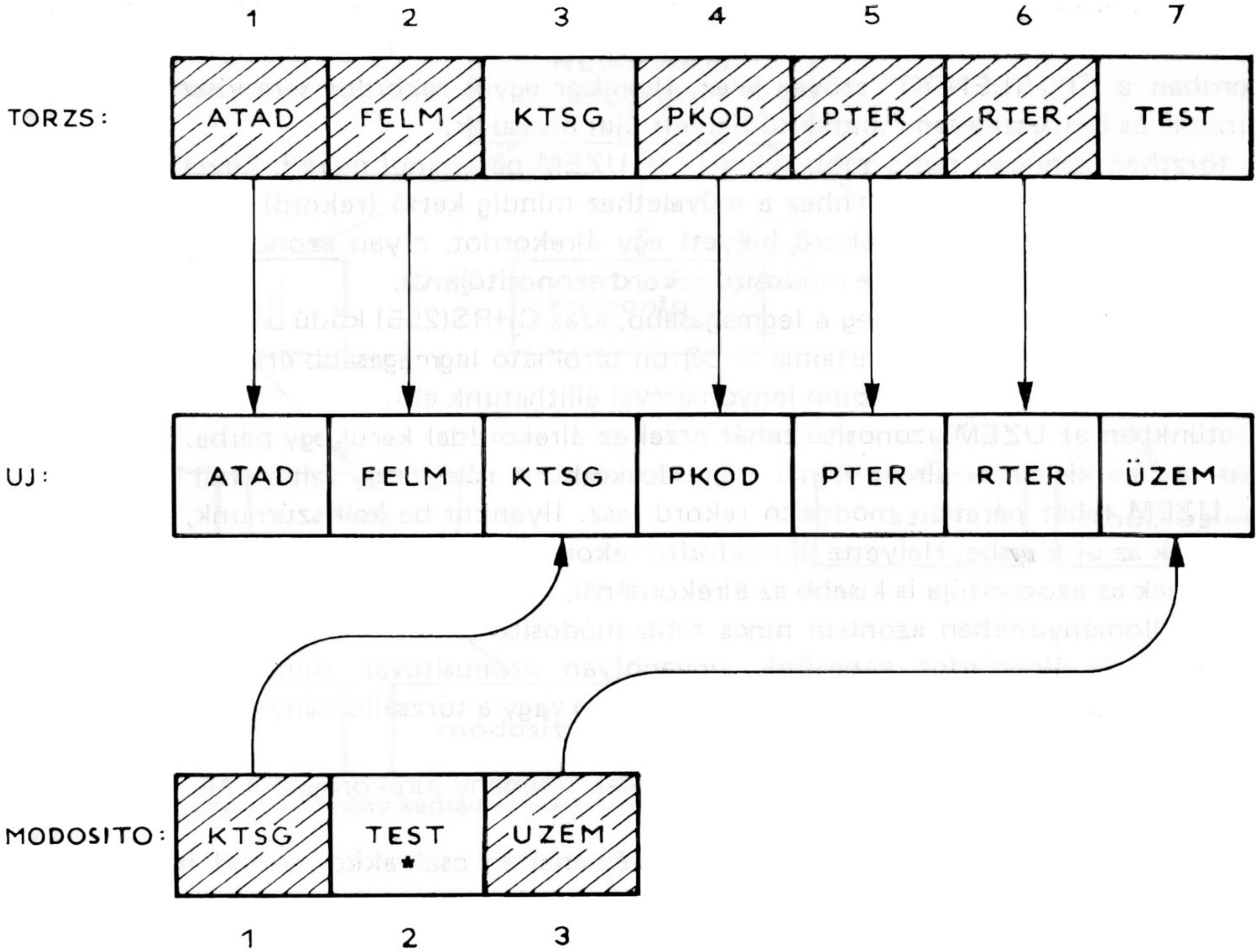
Az állomány végére pedig csak akkor jutunk el, ha a keresett rekord azonosítója, például az UZEM, nagyobb az utolsó rekord azonosítójánál is.

A példában bemutatott kereséseket az állomány egyetlen végigolvasásával, azaz 7 rekord beolvasásával végrehajthattuk, míg a hasonló keresésekhez a rendezetlen állományból 15 rekordot kellene beolvasnunk, keresésenként elölről kezdve az állomány olvasását.

PÁROSÍTÁS

A fenti keresési módot a rendezett állomány karbantartásához (bővítéséhez, szűkítéséhez, módosításához) szükséges párosításhoz is kihasználhatjuk (21. ábra).

Itt a törzsállomány és az ugyanolyan rendezettségű módosító állomány soron következő rekordjaiból képezünk egy párt. E pár tagjai vagy összetartoznak, vagy nem.



21. ábra. Rendezett állomány karbantartása

Az összetartozás úgy dönthető el, hogy összehasonlítjuk az azonosítójukat.
Az első ilyen pár az ATAD és a KTSG azonosítójú törzs-, illetve módosító rekordból áll. Ezek nem tartoznak össze.
Mivel a KTSG azonosító nagyobb az ATAD-nál, már biztos, hogy az ATAD-nak nem is lesz párja. Így az ATAD egy páratlan törzsrekord, és mint ilyet, változtatás nélkül felviszszük az új törzsbe.

Ekkor azonban a pár csonka lesz; a felhasznált tagját, a törzsrekordot pótolnunk kell. Vagyis beolvassuk a következőt.
Most a FELM–KTSG párt vizsgáljuk meg az összetartozás szempontjából. Látható, hogy amíg a törzsrekord azonosítója el nem éri a módosító rekordét, mindig páratlan törzsrekordokkal lesz dolgunk.

Ha elérjük a törzsben a KTSG rekordot, egy összetartozó pár áll előttünk. Ekkor végre kell hajtani a módosítást. Ha csak rekordonkénti módosítást engedünk meg, akkor a módosítás azt jelenti, hogy a módosító KTSG rekordot kell a törzsbeli KTSG rekord helyett felvinnünk az új törzsbe. (Vagyis ez a módosítás voltaképpen csere.)
Ilyenkor a pár mindkét tagját felhasználtuk. Helyükre mind a törzsből, mind a módosító állományból új rekordot kell olvasnunk. Ezek lehetnek ismét összetartozó párok. Esetünkben a PKOD–TEST pár nem ilyen.

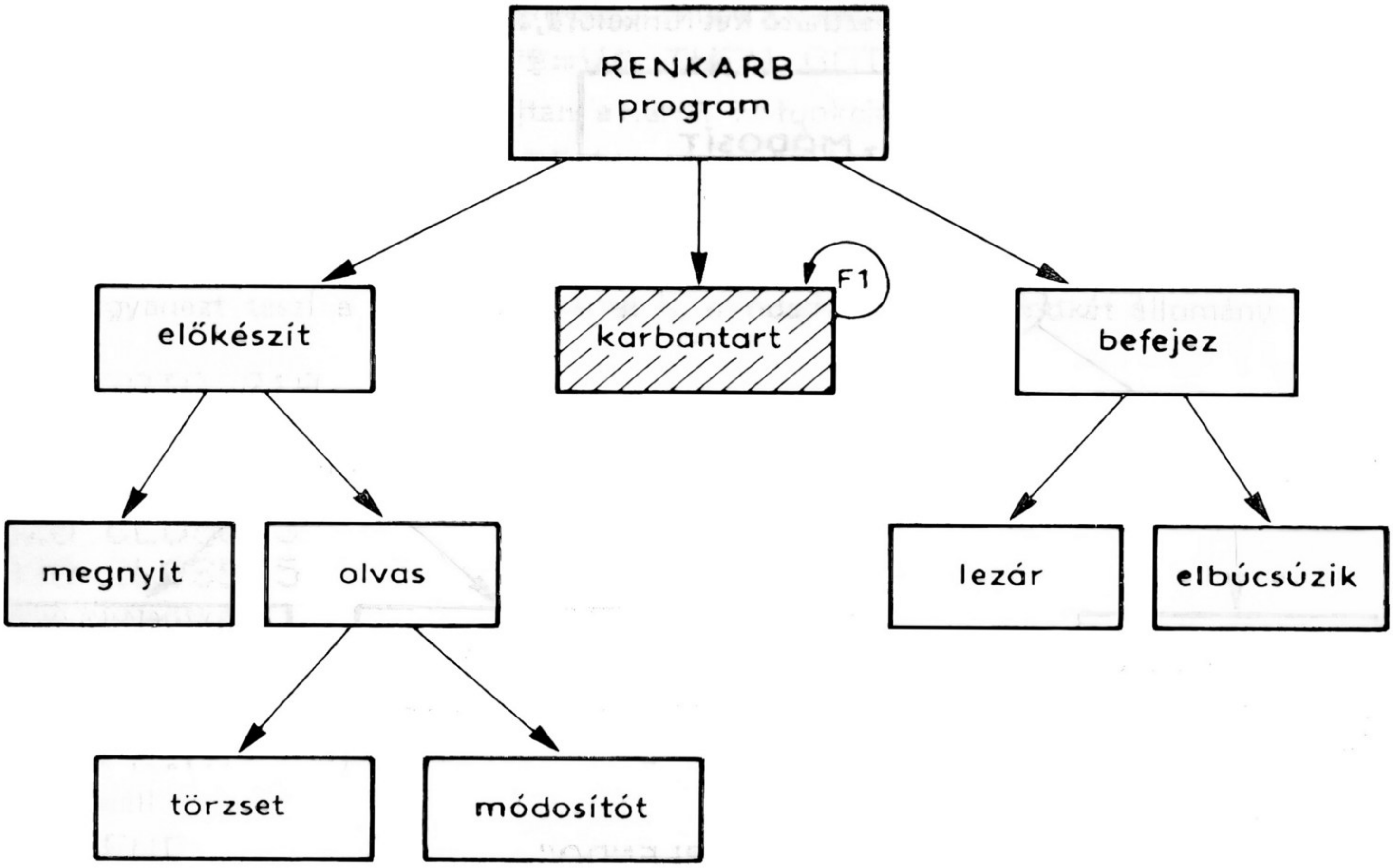
A páratlan törzsrekordokat megint csak felvisszük, amíg el nem érjük a TEST–TEST párt. Itt most nem módosításról, hanem törlésről van szó, amit az ábrán csillag, a tényleges rekordban a "TORLENDO" szöveg jelez. Ilyenkor egyik rekordot sem visszük fel az új törzsbe, és természetesen mindkettő helyett újat olvasunk.
A törzsben azonban nincs több rekord, az UZEM páratlanul marad. Egyetlen rekordot pedig nem lehet párosítani; ehhez a művelethez mindig kettő (rekord) kell. Ezért képeznünk kell a hiányzó törzsrekord helyett egy álrekordot, olyan azonosítóval, amely nagyobb az összes törzs-, illetve módosító rekord azonosítójánál.
Erre a célra természetszerűleg a legmagasabb, azaz CHR$(255) kódú bájtot célszerű használni. Ennek a bájtnak a tartalma a bájton tárolható legmagasabb érték, vagyis a hexadecimális FF lesz, amit a π gomb lenyomásával állíthatunk elő.
Esetünkben az UZEM azonosító tehát ezzel az álrekorddal kerül egy párba. Minthogy az azonosítója kisebb az álrekordénál – gondoskodtunk róla, hogy feltétlenül így legyen –, az UZEM tehát páratlan módosító rekord lesz. Ilyenkor be kell szúrnunk, azaz fel kell vinnünk az új törzsbe. Helyette új módosító rekordot olvasunk.
Ha ennek az azonosítója is kisebb az álrekordénál, ugyanígy járunk el: beszúrjuk.
A mi állományunkban azonban nincs több módosító rekord. Ilyenkor a módosító állományhoz is álrekordot képezünk, ugyanolyan azonosítóval, mint a törzs álrekordjáé volt. Hiszen nem tudhatjuk, hogy a módosító vagy a törzsállomány fog-e hamarabb elfogyni.
A párosítás akkor fejeződik be, ha a pár a két állomány álrekordjaiból áll, azaz ha elértük mindkét állomány végét.
Ebből következik, hogy cserét vagy törlést végrehajtani csak akkor szabad, ha a pár tagjai nem azért tartoznak össze, mert mindketten álrekordok.

KARBANTARTÁS

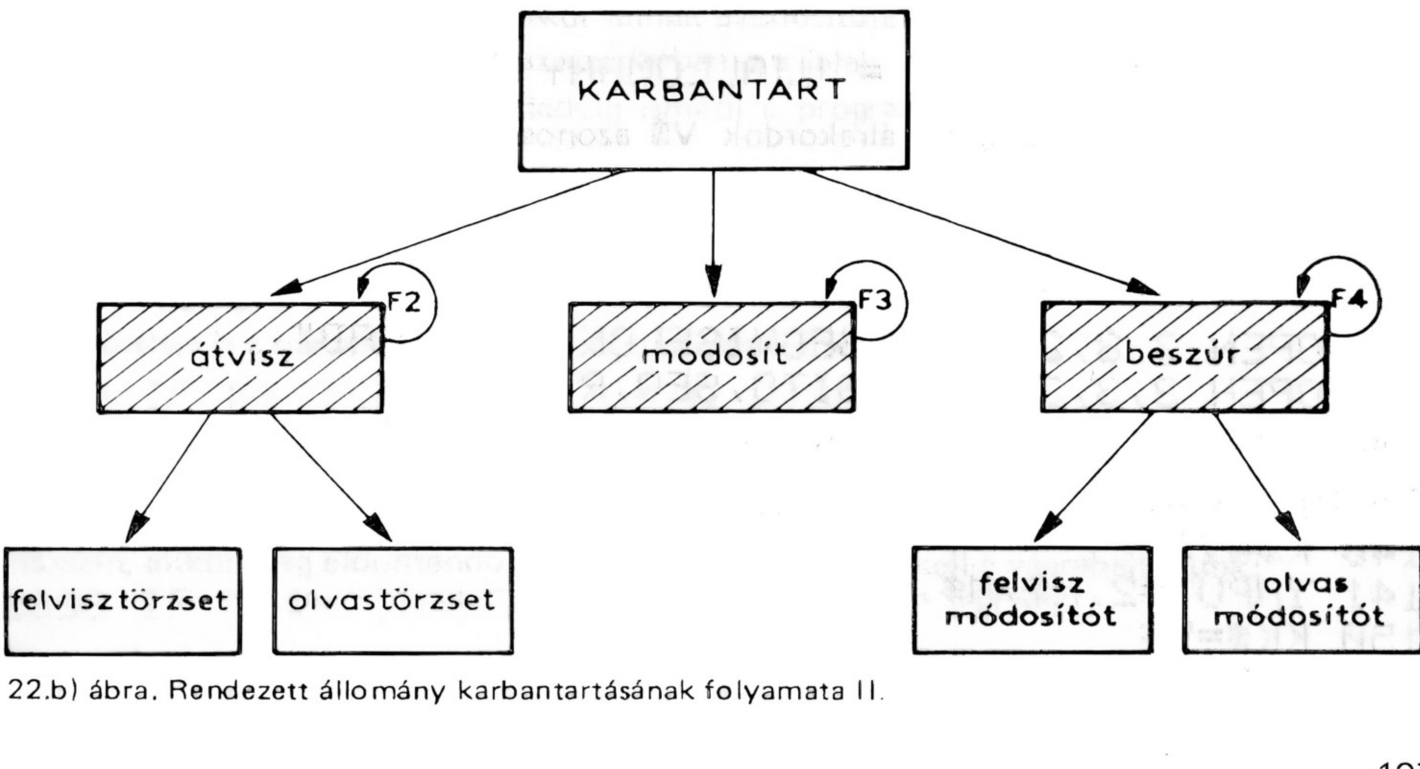
Ezzel a párosítási móddal a rendezett állományok karbantartása rendkívül leegyszerűsíthető, és gazdaságossá tehető.
Ugyanis míg a rendezetlen állományoknál a bővítést, a szűkítést és a módosítást külön menetben kellett végrehajtanunk, és ezek közül legfeljebb az utóbbi kettő vonható össze, itt mind a beszúrást, mind a törlést, mind a cserét egyetlen menetben végezhetjük el. Ehhez csupán a felvivő programmal egy MODOSITO állományt kell létrehoznunk – 22.a) ábra.

Az ábra szerint:
F1: ha még bármelyik állományban van adat, azaz vagy a K$, vagy a KK$ nem négy π jelet tartalmaz.
Megjegyezzük, hogy a K$ a törzsrekord azonosítója, a KK$ pedig a módosító rekordé.
A karbantartás fő tevékenysége, a változatlanul hagyás, a beszúrás és a módosítás, illetve az utóbbin belül a törlés és a csere így jól elkülönül, és mindegyik addig ismétlődik, amíg a rá vonatkozó párosítási feltétel fennáll – 22.b) ábra.

A párosítási feltételek:
F2: ha törzsrekordnak nincs párja, azaz K$ kisebb mint KK$
F3: ha a törzsrekordnak van párja, azaz K$ = KK$, de a pár nem az álrekordokból áll, azaz K$ és KK$ nem π jeleket tartalmaz.
F4: ha a módosító rekordnak nincs párja, azaz K$ nagyobb mint KK$

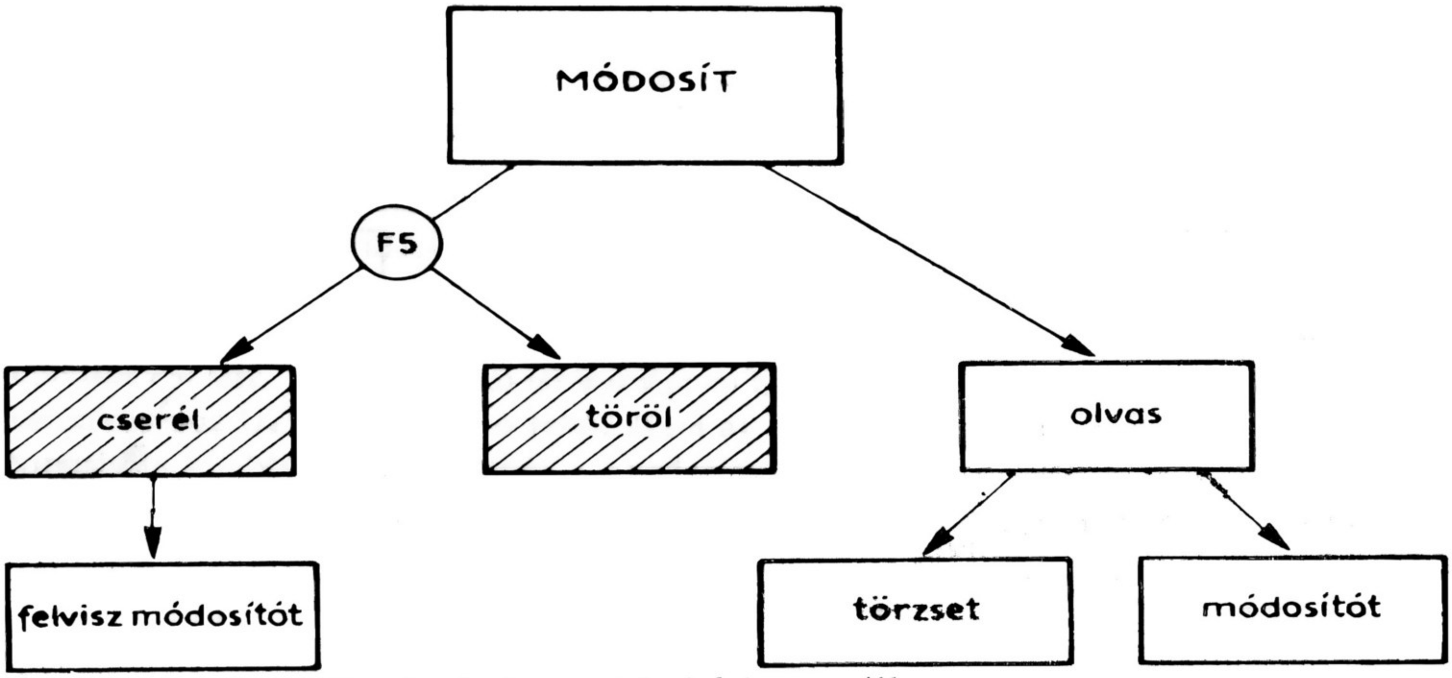


22.a) ábra. Rendezett állomány karbantartásának folyamata I.



22.b) ábra. Rendezett állomány karbantartásának folyamata II.

Maga a módosítás is könnyedén osztható két funkcióra, a cserére és a törlésre — 22.c) ábra.



22.c) ábra. Rendezett állomány karbantartásának folyamata III.

A két funkciót szétválasztó feltétel:
F5: ha a rekord törlendő, azaz MM$ = "TORLENDO"
Most lássuk a KOLTSEGVISELOK állomány rendezett karbantartó programját, amely, mint már megszoktuk, a bejelentkezéssel kezdődik:

```
1 PRINT "♥" : PRINT
2 PRINT " RENDEZETT KARBANTARTAS"
3 PRINT " ———————————————————————"
4 PRINT : PRINT
5 PRINT " TORZS     = KOLTSEGVISELOK"
6 PRINT
7 PRINT " MODOSITO = MODOSITO"
8 PRINT
9 PRINT " UJ        = UJALLOMANY"
```

Beállítja a kezdőértékeket, az álrekordok V$ azonosítóját és az S$ szeparátorjelet:

```
10 V$="ππππ"
11 S$=CHR$(13)
```

Megnyitja az állományokat:

```
110 OPEN 2,8,2,"KOLTSEGVISELOK,SEQ,READ"
120 OPEN 3,8,3,"MODOSITO,SEQ,READ"
130 OPEN 5,8,5,"UJALLOMANY,SEQ,WRITE"
```

Előállítja az első párt:

```
140 K$=V$
141 INPUT#2,K$,M$,T,R
150 KK$=V$
151 INPUT#3,KK$,MM$,TT,RR
```

Megvizsgálja, hogy mindkét állománynak elérte-e a végét:

```
210 IF K$=V$ AND KK$=V$ THEN GOTO 310
```

Ha nem, sorra megkíséreli végrehajtani a három fő funkciót az aktuális párra:

```
220 :   GOSUB 1010 :REM: --> ATVITEL
230 :   GOSUB 2010 :REM: --> MODOSITAS
240 :   GOSUB 3010 :REM: --> BESZURAS
```

Majd ugyanezt teszi a következő párral is, mindaddig, amíg mindkét állomány végére nem ér:

```
299 GOTO 210
```

Ha az állományok elfogytak, lezárja őket, és persze az újat is:

```
310 CLOSE 2
320 CLOSE 3
330 CLOSE 5
```

Majd kijelentkezik:

```
410 PRINT : PRINT
420 PRINT " ------------------------"
430 PRINT " KARBANTARTAS ***KESZ***"
```

Végül leáll:

```
999 END
```

Mindhárom fő funkció végrehajtásáért külön szubrutin felelős.

A változatlanul hagyás csak akkor hajtandó végre, ha a törzsrekord páratlan:

```
1010 IF K$>=KK$ THEN GOTO 1999
```

Ez esetben a program felviszi a törzsrekordot, majd újat olvas helyette:

```
1020 PRINT#5,K$;S$;M$;S$;T;S$;R
1030 K$=V$
1040 INPUT#2,K$,M$,T,R
```

Látható, hogy az álrekordot úgy állítjuk elő, hogy a rekordazonosítót beállítjuk a π jelekre, és ha jött be új rekord, akkor annak azonosítója ezt felülírja, míg az állomány végének elérése után maradnak az azonosítóban a π jelek.

Mindezeket a műveleteket mindaddig ismétli a program, amíg a párosítási feltétel meg nem változik:

```
1099 GOTO 1010
```

Ha a feltétel átáll, a szubrutin visszaadja a vezérlést a hívó programrésznek:

```
1999 RETURN
```

A módosítás is a párosítási feltétel megvizsgálásával kezdődik:

```
2010 IF K$<>KK$ THEN GOTO 2999
```

Ezután azonban azt is meg kell nézni, hogy a pár nem álrekordokból áll-e:

```
2020 IF K$=V$    THEN GOTO 2999
```

Ha nem, akkor még eldöntendő, hogy cserét vagy törlést kell-e végrehajtanunk:

```
2030 IF MM$="TORLENDO" THEN GOTO 2050
```

Ha cseréről van szó, akkor a program felviszi a módosító rekordot az új állományba:

```
2040 PRINT#5,KK$;S$;MM$;S$;TT;S$;RR
```

Akár csere volt, akár törlés, új törzsrekordot és új módosító rekordot olvas:

```
2050 K$=V$
2060 INPUT#2,K$,M$,T,R
2070 KK$=V$
2080 INPUT#3,KK$,MM$,TT,RR
```

A módosítást mindaddig ismétli, amíg a párosítási feltétel fennáll. Annak megváltoztatásakor visszaadja a vezérlést:

```
2099 GOTO 2010
2999 RETURN
```

Látható, hogy a törlés esetén semmilyen tevékenységet nem hajt végre az új rekordok olvasása előtt.

Megjegyezzük, hogy ez nem szokásos; a törölt rekordokat általában kimentjük egy tartalék állományba, vagy legalábbis kilistázzuk. Két okból tesszük ezt: egyrészt fontos, hogy a feldolgozás ne nyeljen el nyomtalanul adatokat, rekordokat, mert akkor nem bizonyíthatjuk a program jóságát; másrészt a tévedésből törölt rekordokat enélkül nehezen tudnánk helyreállítani.

A beszúrás hasonló a változatlanul hagyáshoz, de itt nem a törzs, hanem a módosító rekord kerül az új állományba:

```
3010 IF K$<=KK$ THEN GOTO 3999
3020 PRINT#5,KK$;S$;MM$;S$;TT;S$;RR
3030 KK$=V$
3040 INPUT#3,KK$,MM$,TT,RR
3099 GOTO 3010
3999 RETURN
```

Ez a program is működőképes. NEW után begépelhetjük, és kipróbálhatjuk.
Kipróbálás előtt azonban mentsük ki a lemezünkre, mondjuk RENKARB néven. Majd állítsuk elő a rendezett KOLTSEGVISELOK és MODOSITO állományt.

FIGYELEM!
Ha a programot rendezetlen állományra hajtjuk végre, akkor hibaállapot nem fog bekövetkezni, de a karbantartás csak akkor adja a kívánt eredményt, ha mindkét bemenő állomány azonos rendezettségű, mivel a párosításnál ezt a tulajdonságot erősen kihasználtuk.

Vannak állományok, amelyek természetszerűleg rendezettek. Például a reggel—délben—este mért, és a méréskor az állományba azonnal felvitt hőmérsékleti adatok a keletkezési módjuk miatt szükségképpen időrendi sorrendben állnak.

A költségviselők állománya nem ilyen. A rendezettségéről külön gondoskodnunk kell. Az egyik ilyen megoldás az, hogy az állományt valamilyen rendező eljárás segítségével rendezzük.
A különféle rendezési eljárásokkal egy egész könyvet meg lehetne tölteni, így válasszuk a másik, az egyszerűbb megoldást: az állományt eleve rendezetten hozzuk létre.
Vagyis először töröljük a lemezen esetleg fent levő rendezetlen állományainkat, majd a SEQFELV programmal állítsuk elő a rendezett KOLTSEGVISELOK, majd a rendezett MODOSITO állományt.

Ügyeljünk arra, hogy ezekben a rekordok ATAD, FELM, KTSG, PKOD, PTER, RTER, TEST, illetve KTSG, TEST, UZEM sorrendben legyenek. *Ismételten felhívjuk a figyelmet arra, hogy a program futásakor nem kapunk semmilyen hibajelzést, ha az állományok rendezettsége nem megfelelő. Látszólag tehát minden rendben lesz.* A rendezési hibák éppen azért olyan veszélyesek, mert a hatásuk ,,csak" az eredményben, a nem megfelelően karbantartott új állomány tartalmában jelenik meg.

Ezután már betölthetjük és lefuttathatjuk a rendezett karbantartó RENKARB programot. Az eredményről úgy győződhetünk meg, hogy vagy a SEQPRNT programmal kilistázzuk, vagy a SEQKERD programmal lekérdezzük az új állományt.

Felhívjuk a figyelmet arra, hogy a RENKARB program feltételezi, hogy egy-egy törzsrekordhoz legfeljebb csak egy módosító rekord tartozhat. Ha mégis több jön be, a módosítást csak az elsőre hajtja végre, az összes többi módosító rekordot új rekordként, beszúrásként viszi fel az új állományba.

Ha ezt a problémát úgy akarjuk áthidalni, hogy a módosítás az azonos kulcsú módosító rekordok közül csak az utolsót vegye figyelembe, akkor a módosító modulunkat így kell átírnunk:

```
2010 IF K$<>KK$ THEN GOTO 2999
2020 IF K$=V$   THEN GOTO 2999
2030 :  K1$=KK$:M1$=MM$:T1=TT:R1=RR
2040 :  :  KK$=V$:INPUT#3,KK$,MM$,T,R
2050 :  IF KK$=K1$ THEN GOTO 2030
2060 :  IF M1$="TORLENDO" THEN GOTO 2080
2070 :  :  PRINT#5,K1$;S$;M1$;S$;T1;S$;R1
2080 :  K$=V$:INPUT#2,K$,M$,T,R
2099 GOTO 2010
2999 RETURN
```

Az eljárás lényege, hogy ha talált egy törzs—módosító párt, akkor nem hajtja végre a módosítást, hanem elkezdi olvasni a soron következő módosító rekordokat, és ezt mindaddig teszi, amíg egy olyat nem talál, amelynek kódja nem egyezik meg a párbeli módosító rekord kódjával; ekkor végrehajtja a módosítást, de nem az utoljára beolvasott módosító rekorddal, hanem az eggyel előbbivel.

Bonyolultabb feladatoknál előfordulhat, hogy ezzel sem elégedhetünk meg; például meg kell engednünk a sorozatos módosítást. Ami azt jelenti, hogy minden módosító rekordot figyelembe kell vennünk: előfordulásuk sorrendjében át kell vezetnünk a törzsrekordba a megfelelő módosítást, és az új állományba a rekord csak az utolsó módosító rekord feldolgozása után kerülhet be, magán viselve az összes módosító rekord hatását. Ennek megoldása egyáltalán nem nehéz, a módosítások átvezetése a törzsrekordba a 2030 és 2040 sorok közé beépített eljárással könnyen megoldható. Azt persze meg kell határoznunk, hogy az egyes módosító tételeknek milyen hatásuk legyen; például ha a tényleges ráfordításra jön be módosító adat, akkor azzal a törzsbeli értéket felül kell írni, vagy inkább hozzá kell adni ahhoz. Újabb kérdés, hogy ilyenkor mit tegyünk a törlő rekordokkal. Az egyik lehetséges megoldás az, hogy ha az utolsó módosító rekord törlést ír elő, akkor nem visszük fel a rekordot az új állományba; ha pedig olyan törlő rekordot találunk, amelyet még követnek módosító rekordok, akkor az őt megelőző módosító rekordokat nem vesszük figyelembe, de az őt követőeket már igen. Más megoldás lehet, hogy az utóbbi

esetben a törlő rekord elérésekor a törzsrekord tartalmát valamilyen egyezményes kezdőértékre (nullára, szóközre stb.) állítjuk be, és a további módosításokat ehhez képest hajtjuk végre.

Számos más megközelítés is lehetséges, de ezek taglalása már túlmutat az elemi ismeretek körén. A bemutatott program azonban alkalmas arra, hogy az olvasó a módosítás funkció átírásával különböző karbantartási változatokat is kipróbálhasson.

Végül még egy kérdést tartunk fontosnak a karbantartással és egyáltalán az ismétlődő tevékenységekkel kapcsolatban.

Amint láttuk, a beszúrás, a módosítás, a (változatlan) átvitel ismétlődő tevékenységek, funkciók. A programban az ismétlődést (ciklust) a funkción belül, azaz a GOSUB-bal hívott szubrutinban szerveztük meg.

Megoldható ez úgy is, hogy maga a hívás ciklikus:

```
220 :   IF K$<KK$                  THEN GOSUB 1010:GOTO 220
230 :   IF K$=KK$ AND K$<>V$ THEN GOSUB 2010:GOTO 230
240 :   IF K$>KK$                  THEN GOSUB 3010:GOTO 240
```

Maguk a modulok ilyenkor egyszerűbbek:

```
1000 REM:
1001 REM: ======================================= ATVITEL
1002 REM:
1010 PRINT#5,K$;S$;M$;S$;T;S$;R
1020 K$=V$  :  INPUT#2,K$,M$,T,R
1999 RETURN
2000 REM:
2001 REM: ======================================= MODOSITAS
2002 REM:
2010 IF MM$="TORLENDO" THEN GOTO 2030
2020 :   PRINT#5,KK$;S$;MM$;S$;TT;S$;RR
2030 K$ =V$  :  INPUT#2,K$ ,M$ ,T ,R
2040 KK$=V$  :  INPUT#3,KK$,MM$,TT,RR
2999 RETURN
3000 REM:
3001 REM: ======================================= BESZURAS
3002 REM:
3010 PRINT#5,KK$;S$;MM$;S$;TT;S$;RR
3020 KK$=V$  :  INPUT#3,KK$,MM$,TT,RR
3999 RETURN
```

A két megoldás az eredményét tekintve azonos. Mégis mi szól az egyik vagy másik mellett, illetve ellen?

A hívott modulon belül szervezett ciklus előnye, hogy — különösen interpreterrel történő végrehajtás esetén — a program gyorsabb, hiszen a hívás egy-egy rekordsorozatra csak egyszer hajtódik végre.

A ciklikus hívás ezzel szemben áttekinthetőbb, maga a hívott modul is egyszerűbb és logikusabb, mert kizárólag a rábízott funkciót hajtja végre.

Mellette szól továbbá, hogy az egzakt strukturált programtervezési rendszerek (például a Jackson-módszer) ezt a megoldást tekintik elfogadhatónak. Általában megjegyezhetjük, hogy a ciklusokat célszerű előnyben részesíteni az alternatívákkal szemben. Olyan programozási nyelvek (pl. a COBOL) esetén, amelyek külön utasítással is rendelkeznek a ciklikus hívás végrehajtására, az embernek eszébe se jut más lehetőséget használni.

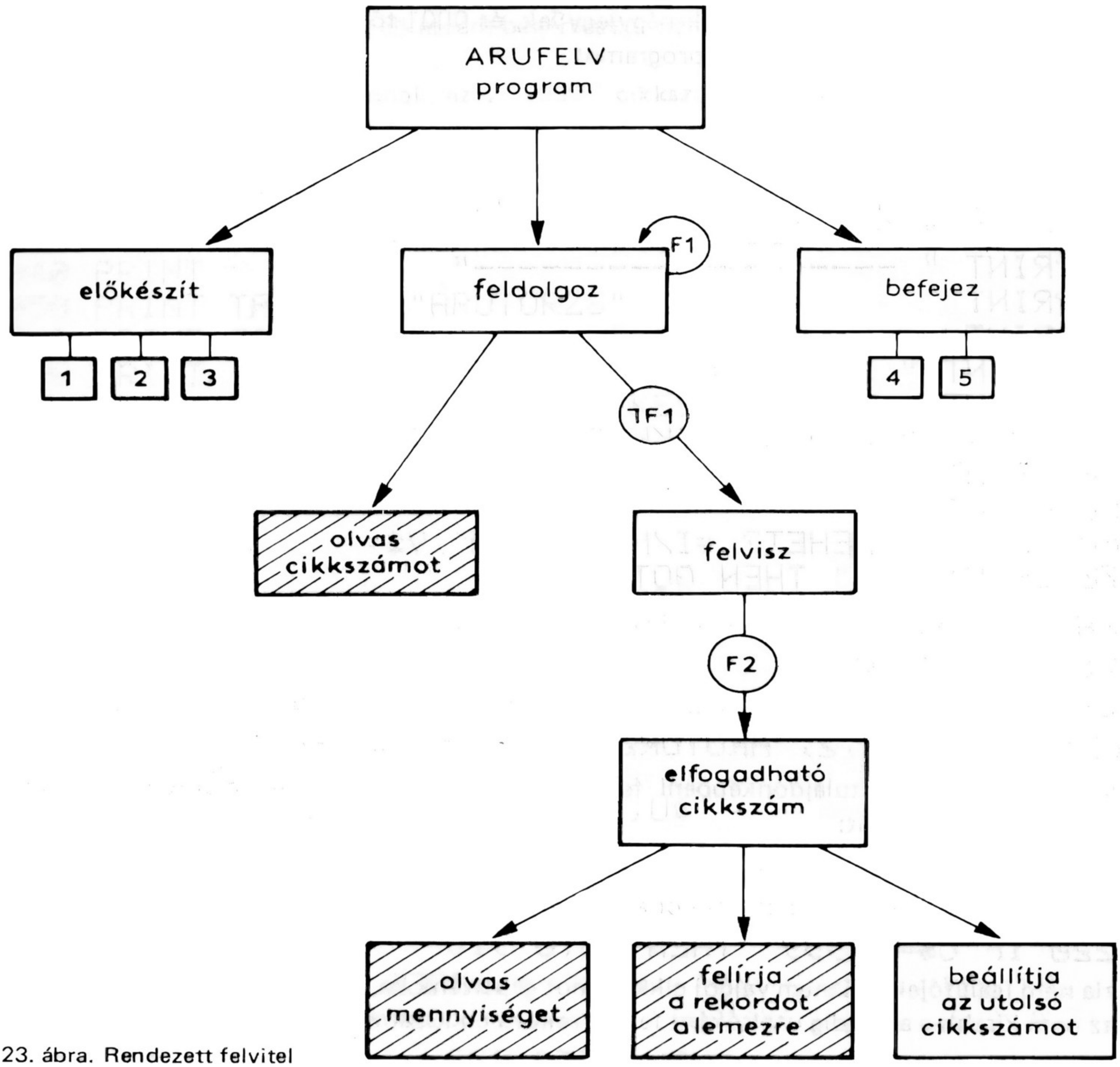
RENDEZETT FELVITEL

A rendezés nemcsak azért költséges, mert időigényes művelet, hanem azért is, mert a végrehajtásához meglehetősen nagy háttértár szükséges. (A korszerű általános rendező eljárások általában a rendezendő állomány lemezterületének legalább a másfélszeresét igényelik ahhoz, hogy a rendezést végre tudják hajtani. Rosszabb esetben még ennél nagyobb lemezterületre is szükségünk lehet – a rendezendő és a rendezett állomány tárolásához szükséges lemezterületen kívül.)

Ezért célszerű a rekordokat eleve rendezetten felvinni az állományba. Ez természetesen nem minden esetben lehetséges, de például már említettük, hogy a napi hőmérsékleti adatok rekordjai időrendi sorrendben keletkeznek. Legalább ilyenkor törekedjünk arra, hogy az adatok felvitelekor a meglevő rendezettséget ne rontsuk el.

A legjobb, ha erről maga a felvivő program gondoskodik, és nem fogad el olyan adatot, amelynek egy már fent levő adat előtt lenne a helye.

Tekintsünk egy egyszerű példát, amelyben már korábban kigyűjtött áruadatokat, az áru



23. ábra. Rendezett felvitel

cikkszámát és az áruból eladott mennyiséget visszük fel egy lemezállományba, további feldolgozás céljából.

Az adatok cikkszám szerint rendezve érkeznek. A felvivő program dolga az, hogy megakadályozza a hibás sorrendben történő begépelést (23. ábra).

Az ábra szerinti feltételek:

F1: ha van még felviendő cikk, azaz a C$ cikkszám nem = "9999",

F2: ha a begépelt C$ cikkszám nem kisebb az utoljára felvitt U$ cikkszámnál

A járulékos (előkészítő és befejező) tevékenységek pedig:

1: a legkisebb cikkszám beállítása

2: bejelentkezés

3: a lemezállomány megnyitása

4: a lemezállomány lezárása

5: kijelentkezés

Megjegyezzük, hogy a cikkszámok négyjegyűek, és 0001-től növekvő sorrendben 9998-ig haladhatnak. Most pedig lássuk a programot.

Első lépése a bejelentkezés:

```
1 PRINT "□"
2 PRINT
3 PRINT " ARUFORGALMI  ADATOK"
4 PRINT " RENDEZETT FELVITELE"
5 PRINT " -------------------"
6 PRINT
7 PRINT
10 PRINT " ELSO   CIKKSZAM = 0000"
20 PRINT
30 PRINT " UTOLSO CIKKSZAM = 9999"
40 PRINT
50 PRINT
60 INPUT " MEHET? =I/N=  I■■■";V$
70 IF V$<>"I" THEN GOTO 999
```

Majd beállítja az utolsó felvitt rekord U$ cikkszámát a lehető legkisebbre:

```
110 U$="0000"
```

Ezután megnyitja a létrehozandó lemezállományt:

```
120 OPEN 2,8,2,"ARUTORZS,SEQ,WRITE"
```

Most következik a tulajdonképpeni feldolgozás. A program bekéri a soron következő rekord C$ cikkszámát:

```
210 GOSUB 1010 :REM: ===> CIKKSZAM
```

Ha annak értéke négy kilences, akkor a felvitelt befejezettnek tekinti:

```
220 IF C$="9999" THEN GOTO 910
```

Ha nem leállítójelet, hanem valódi cikkszámot gépeltünk be, a program megvizsgálja, hogy az nem kisebb-e az eddig utolsóként felvitt rekord cikkszámánál:

```
230 IF C$<U$        THEN GOTO 299
```

Ide építettük be tehát a sorrend elrontása elleni védekezést. A rossz sorrendben begépelt cikkszámot a program figyelmen kívül hagyja.
Ha a cikkszám megfelelő volt, akkor következik a mennyiség bekérése:

```
240 GOSUB 2010 :REM: ===> MENNYISEG
```

Ezután a program a rekordot felviszi a lemezre:

```
250 PRINT#2,C$
260 PRINT#2,M
```

A sikeres felvitel után azonban meg kell változtatni az utolsó felvitt rekordot megjelölő U$ cikkszám értékét, mert most már az éppen felvitt C$ rekord az utolsó:

```
270 U$=C$
```

Végül a program áttér a következő rekord kezelésére:

```
299 GOTO 210
```

Mellesleg ugyanezt teszi, ha a cikkszám bekérésekor nem megfelelő sorrendben következő cikkszámot kapott.
Ha viszont vége a feldolgozásnak, azaz "9999" cikkszámot adtunk meg, akkor lezárja a lemezállományt és kijelentkezik:

```
910 CLOSE 2
920 PRINT "□" : PRINT
930 PRINT TAB(17);"AZ"
940 PRINT
950 PRINT TAB(14);"ARUTORZS"
960 PRINT TAB(14);"█████████"
970 PRINT
980 PRINT TAB(12);"ALLOMANY KESZ"
990 PRINT SPC(250);SPC(250)
```

Végül leáll:

```
999 END
```

A C$ cikkszám bekérését külön szubrutin hajtja végre:

```
1010 PRINT "□"
1020 PRINT
1030 PRINT " ARUFORGALMI ADATOK"
1040 PRINT " ------------------"
1050 PRINT
1060 PRINT " UTOLSO CIKK : ";U$
1070 PRINT
1080 PRINT " ..................."
1090 PRINT
1100 PRINT
1110 INPUT " CIKKSZAM = ";C$
1120 C$=RIGHT$(C$,4)
1999 RETURN
```

Ugyancsak külön szubrutin kéri be az áruból eladott M mennyiséget:

```
2010 PRINT
2020 PRINT
2030 INPUT " MENNYISEG = ";M
2099 RETURN
```

Amint látható, egy rekord beolvasása, azaz az adatbevitel ketté van vágva. Azért tettük ezt, mert a C$ cikkszám vizsgálatától függ, hogy kell-e mennyiséget is beolvasni, avagy sem.

Ezenkívül így a megfelelő adatbekérő szubrutinokban könnyen elhelyezhető az adat hibavizsgálata, ami jelenleg nincs beépítve.

A programot begépelés után mentsük ki a lemezünkre, mondjuk ARUFELV néven. Majd próbáljuk is ki.

Javasoljuk az alábbi próbaadatok felvitelét, mert ezzel az állománnyal a (következő) csoportonkénti feldolgozást végrehajtó programunk is működni fog:

```
ARUFORGALMI  ADATOK
RENDEZETT FELVITELE

ELSO    CIKKSZAM = 0000
UTOLSO  CIKKSZAM = 9999

MEHET? =I/N=? I


ARUFORGALMI ADATOK

UTOLSO CIKK : 0000
********************
CIKKSZAM   = ? 1110

MENNYISEG = ? 100


ARUFORGALMI ADATOK

UTOLSO CIKK : 1110
********************
CIKKSZAM   = ? 1120

MENNYISEG = ? 200


ARUFORGALMI ADATOK

UTOLSO CIKK : 1120
********************
CIKKSZAM   = ? 1130

MENNYISEG = ? 300


ARUFORGALMI ADATOK

UTOLSO CIKK : 1130
********************
CIKKSZAM   = ? 1140

MENNYISEG = ? 400


ARUFORGALMI ADATOK

UTOLSO CIKK : 1140
********************
CIKKSZAM   = ? 1240

MENNYISEG = ? 500
```

```
ARUFORGALMI ADATOK
------------------
UTOLSO CIKK : 1240
* * * * * * * * * * * * * * * * * *
CIKKSZAM   = ? 1260
MENNYISEG = ? 600
```

```
ARUFORGALMI ADATOK
------------------
UTOLSO CIKK : 1260
* * * * * * * * * * * * * * * * * *
CIKKSZAM   = ? 1310
MENNYISEG = ? 700
```

```
ARUFORGALMI ADATOK
------------------
UTOLSO CIKK : 1310
* * * * * * * * * * * * * * * * * *
CIKKSZAM   = ? 1320
MENNYISEG = ? 800
```

```
ARUFORGALMI ADATOK
------------------
UTOLSO CIKK : 1320
* * * * * * * * * * * * * * * * * *
CIKKSZAM   = ? 1330
MENNYISEG = ? 900
```

```
ARUFORGALMI ADATOK
------------------
UTOLSO CIKK : 1330
* * * * * * * * * * * * * * * * * *
CIKKSZAM   = ? 2010
MENNYISEG = ? 800
```

```
ARUFORGALMI ADATOK
------------------
UTOLSO CIKK : 2010
* * * * * * * * * * * * * * * * * *
CIKKSZAM   = ? 2020
MENNYISEG = ? 700
```

```
ARUFORGALMI ADATOK
------------------
UTOLSO CIKK : 2020
* * * * * * * * * * * * * * * * * *
CIKKSZAM   = ? 2120
MENNYISEG = ? 600
```

```
      AZ
   ARUTORZS
   ========
ALLOMANY KESZ
```

FIGYELEM!

Az adatokat hibátlanul kell begépelnünk, mert a programba nincs beépítve adatellenőrzés. Az adatbevitelt "9999" cikkszámmal kell befejezni.

Sorrendi hibákat természetesen véthetünk, sőt esetenként tegyük is meg, hogy lássuk: a program valóban nem engedi meg a hibás sorrendben megadott adatok felvitelét.

Arra persze ügyeljünk, hogy ne felejtsünk ki rekordot, mert a rendezett felvitel miatt ez utólag már nem vihető fel. Az ilyen rekordokat csak karbantartással lehet az állományba beszúrni. Ugyancsak karbantartás útján lehet az esetleg hibásan lemezre került adatokat javítani.
Minthogy az állományhoz nincs karbantartó programunk, a fenti hibák bármelyike esetén az állományt törölnünk, majd újragenerálnunk kell, most már remélhetőleg hibátlanul. Ehelyett természetesen megtehetjük azt is, hogy a RENKARB program mintájára írunk egy karbantartó programot az ARUTORZS-höz.

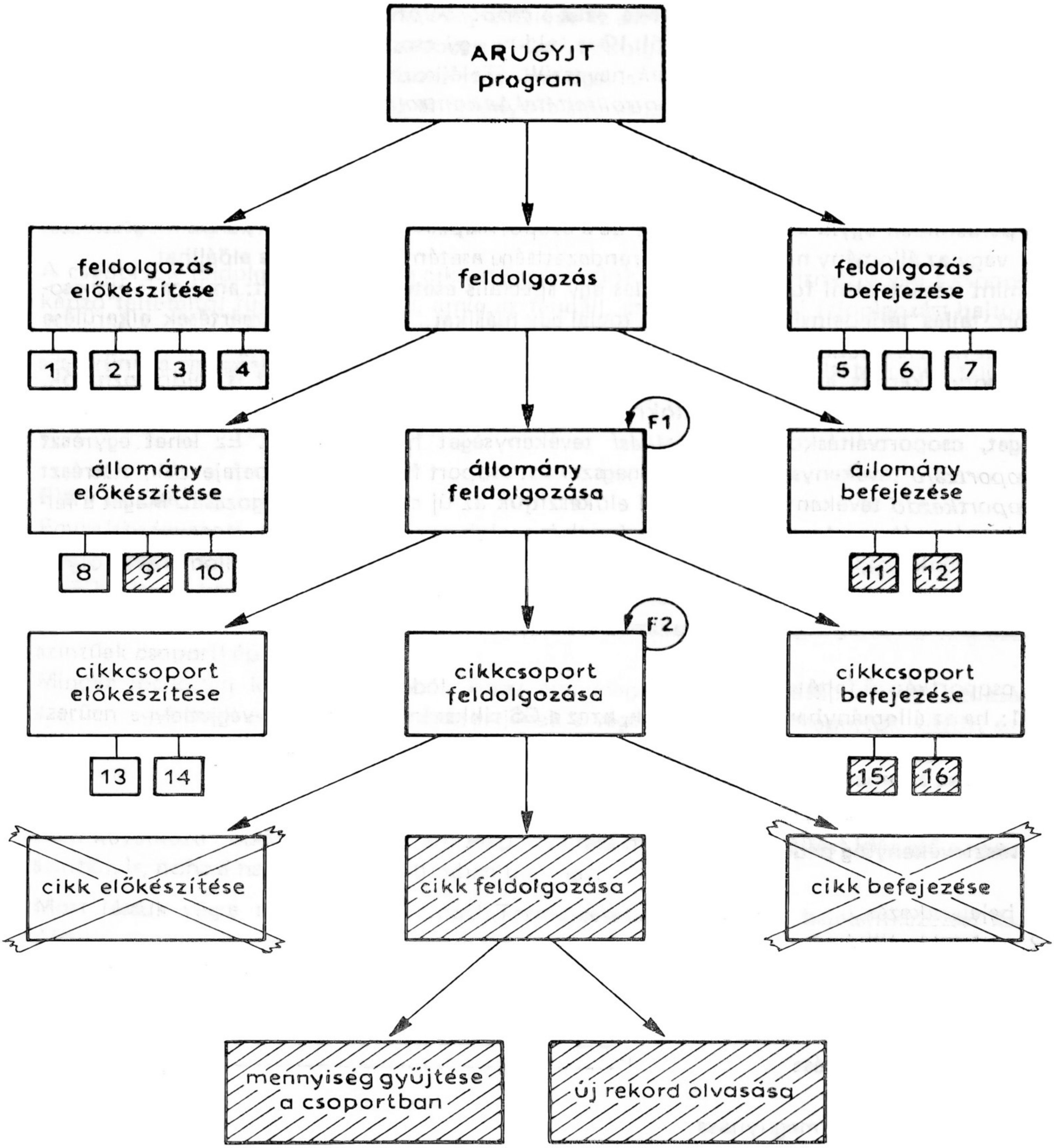
FIGYELEM!
A rendezett felvitel nem jelenti a rekordok rendezését. Feltételezi, hogy azok rendezetten érkeznek, vagy mert eleve így keletkeznek, vagy mert egy program így állítja őket elő, vagy mert a felvitel előtt manuálisan rendeztük őket. A rendezett felvitel csupán arra ügyel, hogy ezt a rendezettséget a felvitel közben ne ronthassuk el.

Kiemeljük a manuális rendezés szerepét. Nyilvánvaló, hogy ez nem jöhet számításba, ha egy törzsállományt kell létrehoznunk, ilyenkor ugyanis általában nagy tömegű adatot kell felvinnünk. Az ilyen felvivő programok ezért többnyire nem figyelik a rendezettséget, a rekordokat mindenképpen felviszik. Az így keletkező „nyers" törzsállományt azután megfelelő rendező eljárással rendezni kell.
A rendezett felvitel akkor juthat szóhoz, ha egy már meglevő rendezett törzsállomány karbantartásához szükséges adatokat kell felvinnünk. Ilyenkor a manuális rendezés sem kizárt, persze csak akkor, ha egy-egy karbantartó menetben nem túl sok rekord szerepel — vagy azért, mert a mozgások száma csekély, vagy azért, mert a karbantartást elegendően sűrűn hajtjuk végre. Néhány tucat rekord kézzel is biztonságosan rendezhető, így nyugodtan használhatjuk a rendezett felvivő programot a karbantartás módosító állományának létrehozására.

CSOPORTONKÉNTI FELDOLGOZÁS

Figyeljünk fel az előbbi programmal előállított árutörzs állomány adataira. A rekordok a cikkszám szerint csoportosíthatók. Tegyük fel, hogy a négyjegyű cikkszám első két jegye egy-egy cikkcsoportot határoz meg. Ekkor azok a cikkek, amelyeknek cikkszáma az első két számjegyben azonos, ugyanahhoz a cikkcsoporthoz tartoznak:

Állo-mány	Cikk-csoportok				
1110	1110				
1120	1120				
1130	1130				
1140	1140				
1240		1240			
1260		1260			
1310			1310		
1320			1320		
1330			1330		
2010				2010	
2020				2020	
2120					2120



24. ábra. Csoportonkénti feldolgozás

Így a teljes állományunk öt csoportra bontható. Azt, hogy valamely rekord melyik csoportba tartozik, a rekord egy adata, a *cikkszám első két jegye* határozza meg. Ezt az adatot *csoportképző* adatnak, az őt tartalmazó *változót* pedig *csoportképző változónak*, vagy röviden *csoportváltozónak* nevezzük. (Elterjedt az eredeti angol kifejezés, a control variable tükörfordítása, a *kontrollváltozó* elnevezés is.)

Amikor a csoportképző változó értéke, azaz a *csoportképző feltétel*, vagy röviden *csoportfeltétel* megváltozik, például 11-ről 12-re, akkor egy csoport befejeződik, és egy új csoport kezdődik. Ezt *csoportváltásnak* nevezzük. (Találkozhatunk a control condition és a control break mintájára képzett *kontrollfeltétel* és *kontrollszakítás* kifejezésekkel is.)
A csoportok között sem átfedés, sem hézag nem megengedett. Az átfedés azt jelenti, hogy valamely adat egyidejűleg több csoportba is tartozik. A hézag ennek ellenkezője; valamely adat egyik csoportnak sem eleme.
A példánkban egyik sem fordul elő, de a csoportképzési feltételek helytelen megválasztása vagy az állomány nem megfelelő rendezettsége esetén mindkét hiba előállhat.
Amint később látni fogjuk, az átfedés egy speciális esete megengedett: amikor is egy csoport teljes terjedelmében magában foglal egy másikat. Ezeket a félreértések elkerülése végett nem is átfedő, hanem egymásba ágyazott csoportoknak nevezzük.
A csoportképzés előnye, hogy az adatok (rekordok) csoportonként is feldolgozhatók. Ilyenkor a csoporton belüli adatokra elvégezzük a rájuk vonatkozó *feldolgozó* tevékenységet, csoportváltáskor pedig *átállási* tevékenységet hajtunk végre. Ez lehet egyrészt *csoportzáró* tevékenység, amikor a megszakadt csoport feldolgozását befejezzük, másrészt *csoportkezdő* tevékenység, amellyel előkészítjük az új csoport feldolgozását. Magát a feldolgozó tevékenységet alaptevékenységnek is szokás nevezni.
Tekintsünk például egy olyan programot, amely a fenti ARUTORZS-állományról olyan kigyűjtött listát készít, amely a cikkcsoportonként összesen eladott, valamint az összes eladott áru mennyiségét tartalmazza (24. ábra).

A csoportképző feltételek
F1: ha az állományban van még cikk, azaz a C$ cikkszám nem = a V$ végjellel
F2: ha a cikk ugyanahhoz a cikkcsoporthoz tartozik, mint a megelőző, azaz E$ = L$, és az állománynak nincs vége, azaz C$ nem = V$

A résztevékenység pedig:

1: bejelentkezés
2: a végjel beállítása
3: a nyomtató megnyitása
4: a fejléc elkészítése
5: a lábléc elkészítése
6: a nyomtató lezárása
7: kijelentkezés
8: a lemezállomány megnyitása
9: az első rekord beolvasása
10: a mindösszesen-gyűjtő kinullázása
11: a mindösszesen-sor kiírása
12: a lemezállomány lezárása
13: a cikkcsoportgyűjtő kinullázása
14: a csoportképző feltétel beállítása a cikkcsoportra
15: a csoport összesen sor kiírása
16: a csoport gyűjtése a mindösszesenbe

A programnak sajátos szerkezete van, amelyet az alakja miatt ,,karácsonyfa'' szerkezetnek hívunk. Általában minden csoportonkénti feldolgozásra ez a jellemző. A karácsonyfának természetesen lehetnek csonka vagy üres ágai is.
Látható továbbá, hogy magát az állományt is egy csoportnak tekinthetjük, amely tartalmazza az összes cikkcsoportot. A csoportok egymásbaágyazása megengedett, ha a magasabb szintű csoport teljes teljedelmében magában foglalja az alacsonyabb szintű csoportokat. Esetünkben ez fennáll, így jogosan jártunk el, amikor az állományt egy ,,végső'' csoportként fogtuk fel.

A csoportok feldolgozására olyan ciklust szerveztünk, amelynek az ismétlődése a csoportképző feltételtől függ. Mindaddig, amíg az fennáll, csoporton belüli feldolgozást hajtunk végre. Csoportváltáskor előbb végrehajtjuk a csoport befejező tevékenységét, majd visszatérünk a magasabb szintű csoportba. Ha annak a csoportképző feltétele még teljesül, akkor új alárendelt csoportot kezdünk, az előkészítő tevékenysége végrehajtásával. Így járunk el minden egyes csoport esetén.

FIGYELEM!

Egymásbaágyazott csoportoknál az alacsonyabb szintű csoport csoportképző feltételének magában kell foglalnia a magasabb szintű csoport csoportképző feltételét is. Vagyis az F2 feltétel két részből áll: a cikkcsoport saját feltételéből (E$=L$), és az F1 feltételből. Úgy is fogalmazhatunk, hogy az alacsonyabb szintű csoportok ,,öröklik'' a magasabb szintűek csoportképző feltételét.
Mindez logikusan következik abból, hogy egy magasabb szintű csoport vége szükségszerűen egybeesik egy alárendelt csoportjának a végével, hiszen a csoportok között sem átfedés, sem hézag nem lehet, és az egymásbaágyazás csak teljes tartalmazással megengedett.
Például képzeljünk el egy év–hó–nap szerinti csoportképzést. Ilyenkor a "850201" után következő "850301" dátum a hónap átváltása miatt csoportváltást idéz elő a napok szintjén is, noha a nap mindkét dátumban ugyanaz.

Most lássuk végre magát a programot! Természetesen az most is bejelentkezéssel kezdődik:

```
1 PRINT "■"
2 PRINT
3 PRINT " FORGALOM OSSZESITES"
4 PRINT " -----------------------"
5 PRINT
6 PRINT
```

Eldönthetjük, hogy a listát nyomtatóra vagy a képernyőre kívánjuk elkészíttetni:

```
7 INPUT " NYOMTATORA? =I/N=  N■■■■";V$
8 E=3 : IF V$="I" THEN E=4
```

Ilyen célra általában a nyomtatót használjuk, de ha nincs nyomtatónk, bátran válaszszuk a képernyőt is — az állományt úgy terveztük, hogy a lista egy képernyőre kiférjen.
A program ezután beállítja a V$ végjelet, és megnyitja a nyomtatót (vagy ha a képer-

nyőt választottuk, akkor a képernyőre kimenő állományt), és elkészíti a fejlécet. Vagyis végrehajtja a feldolgozás előkészítését:

```
100 V$="9999"
110 OPEN 1,E
120 PRINT#1,"□"
130 PRINT#1
140 PRINT#1," FORGALOM OSSZESITO"
150 PRINT#1," =================="
160 PRINT#1
```

Most következik maga a feldolgozás. Ezen belül a program megnyitja a lemezállományt, beolvassa az első rekordot, és kinullázza a T mindösszesen gyűjtőt. Ezzel előkészíti az állomány, azaz a legfelső szintű csoport feldolgozását:

```
210 OPEN 2,8,2,"ARUTORZS,SEQ,READ"
220 C$=V$
230 INPUT#2,C$,M
240 L$=LEFT$(C$,2)
250 T=0
```

Ha a csoportképző feltétel fennáll, azaz ha az állománynak nincs vége, megkezdi az állomány feldolgozását:

```
310 IF C$=V$ THEN GOTO 610
```

Ezen belül kinullázza a cikkcsoport C gyűjtőjét, és beállítja a csoportképző feltételét, azaz kimenti a cikkszám első két L$ jegyét az E$ csoportképző változóba. Vagyis előkészíti az alárendelt csoport, a cikkcsoport feldolgozását:

```
320 :   C=0
330 :   E$=L$
```

Ha a csoportképző feltétel fennáll, vagyis ha az aktuális cikkcsoport első két L$ jegye megegyezik az E$ kimentettel, és az állománynak sincs vége, akkor megkezdi az alárendelt csoport, az aktuális cikkcsoport feldolgozását:

```
410 :   IF E$<>L$ OR C$=V$ THEN GOTO 510
```

Ezen belül gyűjti az M eladott mennyiséget a cikkcsoport C gyűjtőjében, majd új rekordot olvas be:

```
420 :   :   C=C+M
430 :   :   C$=V$
440 :   :   INPUT#2,C$,M
450 :   :   L$=LEFT$(C$,2)
```

Ezután folytatja a cikkcsoport feldolgozását:

```
499 :   GOTO 410
```

Ha az újonnan beolvasott rekord már nem az aktuális cikkcsoporthoz tartozik, azaz csoportváltás következett be, akkor befejezi a csoport feldolgozását, azaz kinyomtatja a cikkcsoport C gyűjtőjét, és gyűjti az abban tárolt összeget a mindösszesenben, azaz az állomány T gyűjtőjében:

```
510 :   M$=RIGHT$("         "+STR$(C),7)
520 :   PRINT#1," ";E$;SPC(7);"= ";M$
530 :   PRINT#1
540 :   T=T+C
```

Minthogy itt vége az alárendelt csoport feldolgozásának, visszatér a magasabb szintű csoport feldolgozására:

```
599 GOTO 310
```

Ha a magasabb szintű csoport is átvált, azaz ha vége az állománynak, a program kinyomtatja a T mindösszesen gyűjtőt, és lezárja a lemezállományt:

```
610 M$=RIGHT$("         "+STR$(T),7)
620 PRINT#1," ---------------------"
630 PRINT#1," OSSZESEN = ";M$
710 CLOSE 2
```

Ezzel az állomány feldolgozásának, azaz a magasabb szintű csoportnak is vége. A program tehát befejezi a feldolgozást, vagyis kinyomtatja a láblécet, lezárja a nyomtatót:

```
810 PRINT#1," ---------------------"
820 CLOSE 1
```

Végül elbúcsúzik:

```
830 IF E=3 THEN GOTO 999
840 :   PRINT
850 :   PRINT
860 :   PRINT " A LISTA KESZ A <";E;"> EGYSEGEN."
```

Majd leáll:

```
999 END
```

Megjegyezzük, hogy az állomány végének figyelését a cikkszámba betöltött végjel felülolvasásával, a már ismert módon oldottuk meg.

A C és T összegek kiírása azért ilyen bonyolult, mert az adatokat oszloposan, azaz helyiérték szerint jobbra zárva kívántuk kinyomtatni.

A begépelt programot mentsük ki ARUGYJT néven, majd ha már létrehoztuk az ARUTORZS állományt, a programot azonnal ki is próbálhatjuk. Akár a nyomtatón, akár a képernyőn, az alábbi listának kell megjelennie:

```
FORGALOM OSSZESITO
==================

11        =    1000

12        =    1100

13        =    2400

20        =    1500
```

```
21          =       600
--------------------------------
ÖSSZESEN    =      6600
--------------------------------
```

A tervezett programszerkezetünk előnye, hogy könnyen módosítható. Ha például nemcsak az összesített eladásokat akarjuk a listán megjeleníteni, hanem minden cikk eladott mennyiségét is, a változtatás egyszerű: a mennyiség gyűjtése (a cikkcsoport gyűjtőjébe) és az új rekord beolvasása közé el kell helyeznünk a cikkszám és a mennyiség kinyomtatását. Az ennek megfelelő PRINT utasítás a 425-ös sorba kódolható. (Ilyenkor persze célszerű az összegsorokat feltűnően kiemelni, például eltérő sortávolsággal vagy pozicionálással, speciális jellel vagy aláhúzással stb., hogy a tételsoroktól a listán jól látható módon elkülönüljenek.)

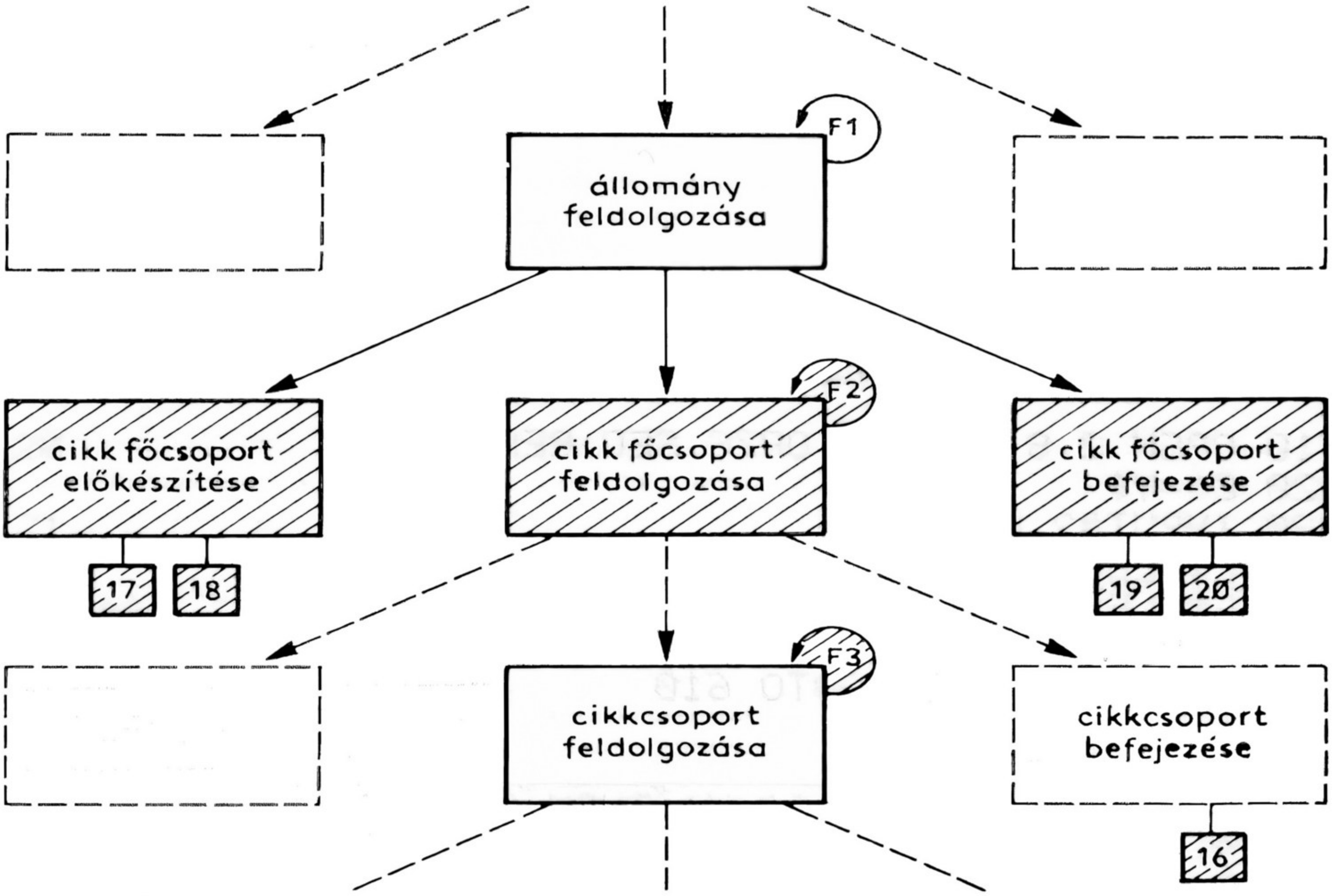
Ennél lényegesen bonyolultabb módosítások is könnyedén végrehajthatók. Tegyük fel például, hogy egy új szintű csoportot képezünk az állomány és a cikkcsoportok között. Vagyis a listát cikkfőcsoportonkénti, és azon belül cikkcsoportonkénti bontásban kívánjuk elkészíteni. A cikkfőcsoportot a cikkszám első jegye határozza meg.

Állomány	Cikk főcsoportok		Cikkcsoportok		
1110	1110		1110		
1120	1120		1120		
1130	1130		1130		
1140	1140		1140		
1240	1240			1240	
1260	1260			1260	
1310	1310				1310
1320	1320				1320
1330	1330				1330
2010		2010	2010		
2020		2020	2020		
2120		2120		2120	

Itt tehát az állományon belül két cikkfőcsoport van, az 1-es és a 2-es. Az első cikkfőcsoport három cikkcsoportot tartalmaz, a 11-est, a 12-est és a 13-ast. A második cikkfőcsoporthoz két cikkcsoport tartozik, a 20-as és a 21-es.
Az új csoportfokozatnak megfelelően új szintet kell beépítenünk a fába (25. ábra).

A módosítás nemcsak az új szintet érinti, hanem kihat az alacsonyabb szintekre is. Az öröklődés miatt elsősorban módosulnak a csoportképző feltételek:

F1: ha nincs vége az állománynak (változatlan)
F2: ha az első jegy = kimentett, és F1 (új feltétel)
F3: ha az első két jegy = kimentett, és F2 (módosult)



25. ábra. Új szint beépítése

Az új szint előkészítő és befejező tevékenységei:

17: a cikkfőcsoport-gyűjtő kinullázása
18: a csoportképző feltétel beállítása a cikkfőcsoportra
19: a cikkfőcsoport összesen sor kiírása
20: a cikkfőcsoport gyűjtése a mindösszesenbe

A régi szinteken a feltételektől eltekintve, csak a cikkcsoport gyűjtése módosul:

16: a cikkcsoport gyűjtése a cikkfőcsoport összesenbe

Ezenkívül minden olvasás után nemcsak az első két karaktert, hanem az első karaktert is le kell választanunk a cikkszámról.
Mindezek a módosítások könnyedén átvezethetők a programba:

```
241 F$=LEFT$(C$,1)
311 F=0
312 G$=F$
313 IF G$<>F$ OR C$=V$ THEN GOTO 550
410 IF E$<>L$ OR G$<>F$ OR C$=V$ THEN GOTO 510
451 F$=LEFT$(C$,1)
540 F=F+C
```

```
549 GOTO 313
550 M$=RIGHT$("       "+STR$(F),7)
560 PRINT#1," ****** ";G$;" = ";M$
570 PRINT#1
580 T=T+F
```

Megjegyezzük, hogy ilyen módosítások esetén a program toldozása-foldozása helyett célszerűbb a teljes érintett részt újraírni. Ezzel nemcsak áttekinthetőbb lesz a program, hanem megbízhatóbb is. (A listán kiemeléssel megjelöltük a módosításokat.)

```
210 OPEN 2,8,2,"ARUTORZS,SEQ,READ"
220 C$=V$
230 INPUT#2,C$,M
240 L$=LEFT$(C$,2)
241 F$=LEFT$(C$,1)
250 T=0
310 IF C$=V$ THEN GOTO 610
311 :  F=0
312 :  G$=F$
313 :  IF G$<>F$ OR C$=V$ THEN GOTO 550
320 :  :  C=0
330 :  :  E$=L$
410 :  :  IF E$<>L$ OR G$<>F$ OR C$=V$ THEN GOTO 510
420 :  :  :  C=C+M
430 :  :  :  C$=V$
440 :  :  :  INPUT#2,C$,M
450 :  :  :  L$=LEFT$(C$,2)
451 :  :  :  F$=LEFT$(C$,1)
499 :  :  GOTO 410
510 :  :  M$=RIGHT$("       "+STR$(C),7)
520 :  :  PRINT#1," ";E$;SPC(7);"= ";M$
530 :  :  PRINT#1
540 :  :  F=F+C
549 :  GOTO 313
550 :  M$=RIGHT$("       "+STR$(F),7)
560 :  PRINT#1," ****** ";G$;" = ";M$
570 :  PRINT#1
580 :  T=T+F
599 GOTO 310
```

Az így átírt ARUGYJT programot nyugodtan kipróbálhatjuk. Minthogy az ARUTORZS állományt csak olvassa, a tévesen végrehajtott módosításaink az adatokat elrontani nem fogják – legfeljebb hibás eredmény jelenik meg. Ha minden rendben van, az alábbi listát kell kapnunk:

```
FORGALOM OSSZESITO
==================

11          =   1000

12          =   1100

13          =   2400

****** 1    =   4500

20          =   1500

21          =    600

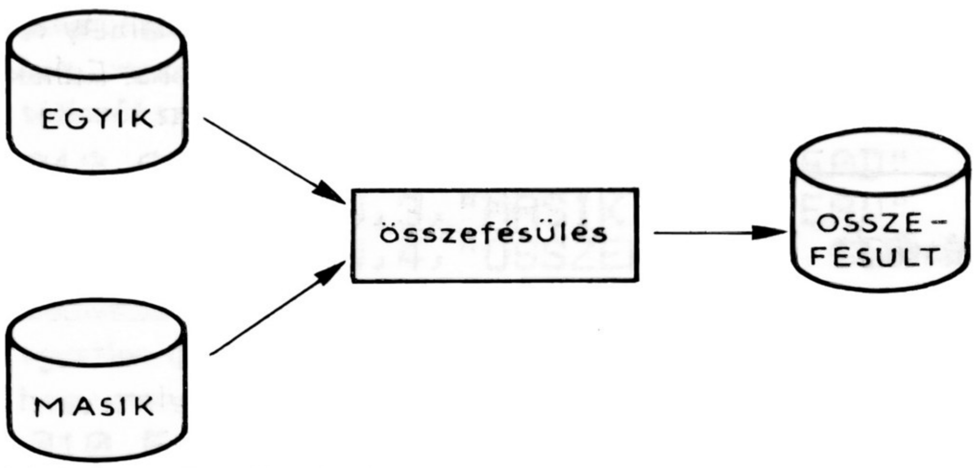
****** 2    =   2100

------------------

OSSZESEN    =   6600
------------------
```

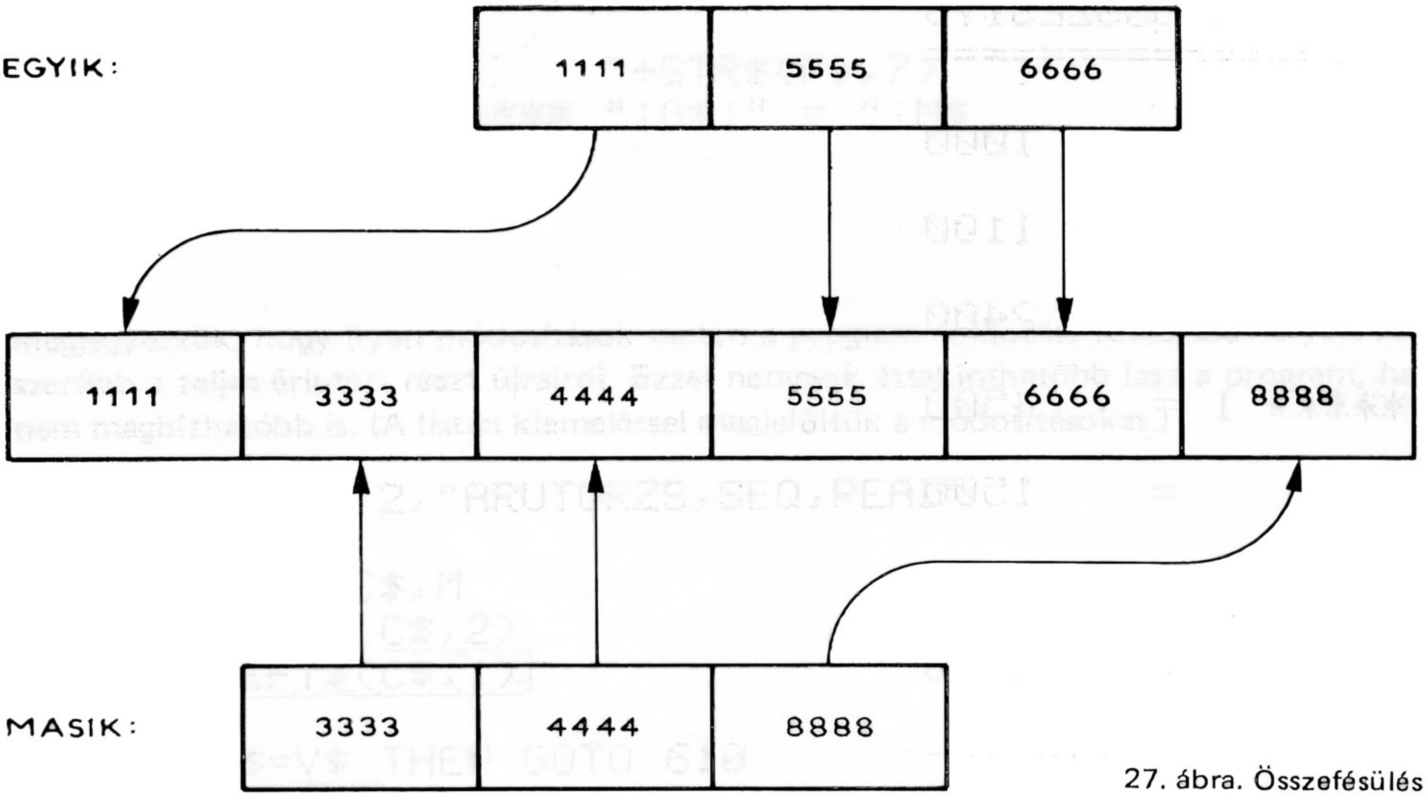
ÖSSZEFÉSÜLÉS

A rendezett állományok feldolgozásának egyik jellegzetes művelete az összefésülés. Ez olyan tevékenység, melynek során két azonos rendeltetésű és rendezettségű állományból egy ugyanolyan rendeltetésű és rendezettségű állományt állítunk elő (26. ábra).



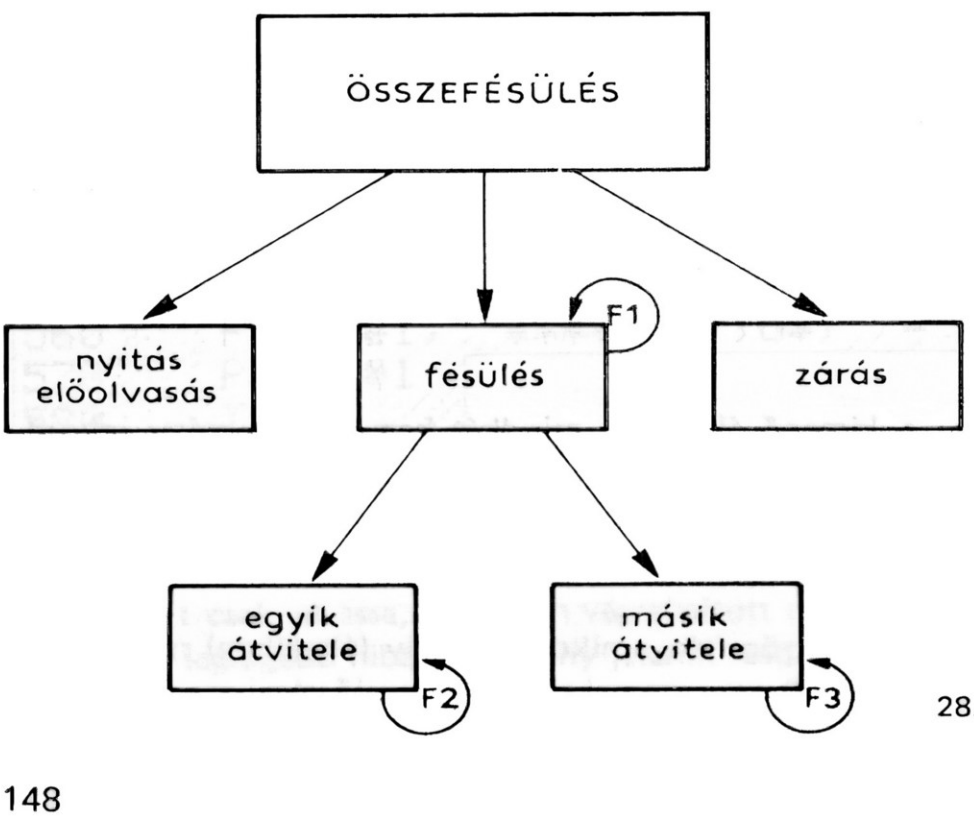
26. ábra. Az összefésülés elve

Az összefésülés lényege, hogy a kimenő állomány mindkét bemenő állomány rekordjait tartalmazza, de úgy, hogy az eredeti rendezettségük együttesen is megmarad (27. ábra).
Az összefésülés igénye általában olyankor merül fel, ha valamely állomány nem egyetlen menetben jön létre; például amikor annak adatai különböző helyen vagy különböző időben keletkeznek.
Ennek tipikus esete a párhuzamos adatrögzítés, amikor valamely (általában) nagy törzsállomány adatait egyidejűleg több felvivő programmal több gépkezelő végzi.



27. ábra. Összefésülés

Az így létrejött állományokat azután vagy összekapcsoljuk, vagy összefésüljük.
Az összekapcsolás (mint a COMMODORE esetén a COPY) gyorsabb, de elrontja a rendezettséget. Ilyenkor még külön rendeznünk is kell az összekapcsolt állományt. (Mindenképpen ez a teendő, ha a felvitel eleve rendezetlen.)
Az összefésülés hosszabb, de megtakarítható a rendezés – feltéve, ha a felvitel rendezett volt, mert rendezetlen állományokat nem lehet összefésülni. (De nem is érdemes.)
Az összefésülés feltételezi, hogy az állományok között nincs átfedés, azaz valamely rekord kizárólag az egyikben vagy a másikban fordulhat elő, mindkettőben soha. Ennek megfelelően szerkezete meglehetősen egyszerű (28. ábra).



28. ábra. Az összefésülés folyamata

Az ábra szerint:

F1: ismétlődik mindaddig, amíg az egyik állományból beolvasott rekord kulcsa meg nem egyezik a másikból beolvasott rekord kulcsával (ami csak akkor lehet, ha mindkettő végjel)

F2: ismétlődik mindaddig, amíg az egyik állományból beolvasott rekord kulcsa kisebb a másikból beolvasott rekordénál

F3: ismétlődik mindaddig, amíg a másik állományból beolvasott rekord kulcsa kisebb az egyikből beolvasott rekordénál

Vegyük észre, hogy az összefésülés voltaképpen a rendezett karbantartás egy speciális esete, amikor is kizárólag beszúrás fordul elő.

A programot a változatosság kedvéért a karbantartástól eltérő módon, ciklikus hívással, paraméterezéssel vezérelt I/O rutinokkal, azaz író/olvasó szubrutinokkal oldjuk meg. A programunk az ARUFELV programmal létrehozott árutörzs állományok összefésülésére alkalmas.

A program bejelentkezéssel indul:

```
110 PRINT "□"
120 PRINT " OSSZEFESULES"
130 PRINT " ------------"
140 PRINT:PRINT " EGYIK+MASIK ==> OSSZEFESULT"
```

Definiálja a V$ végjelet (high value), és egy-egy tömböt a C$ cikkszámnak és az M mennyiségnek. Ezekre csak azért van szükség, hogy kényelmesen paraméterezhessük az olvasó rutint:

```
150 V$="ππππ"
160 DIM C$(3),M(3)
```

Megnyitja az állományokat. Látható, hogy kötött állománynevekkel dolgozik. Ha óhajtjuk, persze átírhatjuk úgy is, hogy bekérje az állományneveket. Erre azonban nincs feltétlenül szükség, a program így is kipróbálható lesz.

```
210 OPEN 2,8,2,"EGYIK,SEQ,READ"
220 OPEN 3,8,3,"MASIK,SEQ,READ"
230 OPEN 4,8,4,"OSSZEFESULT,SEQ,WRITE"
```

Beolvassa az első rekordpárt. Minthogy azonos rekordszerkezetű állományaink vannak, egyetlen olvasó rutin is elég. Az F állományazonosítóval paraméterként határozzuk meg, hogy melyik állományból olvasson:

```
310 F=2:GOSUB 1010
320 F=3:GOSUB 1010
```

A program felváltva visz fel rekordsorozatokat hol az egyik, hol a másik állományból az új állományba, attól függően, hogy melyik bemenő állomány rekordjainak C$ azonosítója a kisebb. Mindezt ismétli mindaddig, amíg a C$ rekordazonosítók azonosak nem lesznek, ami csak akkor állhat elő, ha mindkettő végjel:

```
410 IF C$(2)=C$(3) THEN GOTO 510
420 :  IF C$(2)<C$(3) THEN F=2:GOSUB 2010:GOTO 420
430 :  IF C$(3)<C$(2) THEN F=3:GOSUB 2010:GOTO 430
499 GOTO 410
```

Ha elérte mindkét bemenő állomány végét, lezárja az állományokat:

```
510 CLOSE 4
520 CLOSE 3
530 CLOSE 2
```

Végül elbúcsúzik és leáll:

```
610 PRINT:PRINT "KESZ"
999 END
```

Az olvasó rutin a lehető legegyszerűbb; beállítja a V$ végjelet a C$ rekordkulcsba, majd olvas. Hogy melyik állományból olvasson, és melyik rekordazonosítóba, illetve mennyiségbe töltse be az adatokat, azt az F állományazonosító határozza meg. (Vagyis az adatokat az állományazonosítóval indexeljük. Így a C$(1) az egyik állomány rekordjának cikkszáma, C$(2) pedig a másiké. Ugyanez vonatkozik az M mennyiségre is.)

```
1010 C$(F)=V$:INPUT#F,C$(F),M(F)
1999 RETURN
```

Az olvasáshoz hasonlóan az író rutin is paraméterrel működik. Az F állományazonosító határozza meg, hogy melyik C$ cikkszámot és melyik mennyiséget kell az új állományba felvinni. Sőt, ugyanez az F jelzi azt is, hogy melyik bemenő állományból kell új rekordot olvasni:

```
2010 PRINT#4,C$(F);CHR$(13);M(F)
2020 PRINT:PRINT C$(F)
2030 GOSUB 1010
2999 RETURN
```

Ebben a rutinban a képernyőre írás csak a program tesztelésének egyszerűsítése végett szerepel.

Íme a teljes program:

```
100 REM:
101 REM: ============================= OSSZEFESULES
102 REM:
110 PRINT "♥"
120 PRINT " OSSZEFESULES"
130 PRINT " ------------"
140 PRINT:PRINT " EGYIK+MASIK ==> OSSZEFESULT"
150 V$="ππππ"
160 DIM C$(3),M(3)
200 REM:
201 REM: ------------------------------ MEGNYITASOK
202 REM:
210 OPEN 2,8,2,"EGYIK,SEQ,READ"
220 OPEN 3,8,3,"MASIK,SEQ,READ"
230 OPEN 4,8,4,"OSSZEFESULT,SEQ,WRITE"
300 REM:
```

```
301 REM: -----------------------------------------ELOOLVASAS
302 REM:
310 F=2:GOSUB 1010
320 F=3:GOSUB 1010
400 REM:
401 REM: ---------------------------------------OSSZEFESULES
402 REM:
410 IF C$(2)=C$(3) THEN GOTO 510
420 :  IF C$(2)<C$(3) THEN F=2:GOSUB 2010:GOTO 420
430 :  IF C$(3)<C$(2) THEN F=3:GOSUB 2010:GOTO 430
499 GOTO 410
500 REM:
501 REM: --------------------------------------------ZARASOK
502 REM:
510 CLOSE 4
520 CLOSE 3
530 CLOSE 2
600 REM:
601 REM: ------------------------------------------BEFEJEZES
602 REM:
610 PRINT:PRINT "KESZ"
999 END
1000 REM:
1001 REM: ==================================OLVASAS
1002 REM:
1010 C$(F)=V$:INPUT#F,C$(F),M(F)
1999 RETURN
2000 REM:
2001 REM: ================================FELVITEL
2002 REM:
2010 PRINT#4,C$(F);CHR$(13);M(F)
2020 PRINT:PRINT C$(F)
2030 GOSUB 1010
2999 RETURN
```

Mentsük ki a programot ARUFESU néven, de még ne próbáljuk ki!
Előbb hozzunk létre az ARUFELV programmal egy EGYIK nevű állományt, 1111, 5555, 6666 kulcsú (cikkszámú) rekordokkal, tetszőleges mennyiségekkel.

FIGYELEM!
Ehhez az ARUFELV program 120-as sorában módosítani kell az OPEN utasítást, kicserélve az ARUTORZS nevet EGYIK-re.

Hasonló módosítással hozzunk létre egy MASIK nevű állományt is, 3333, 4444, 8888 kulcsú (cikkszámú) rekordokkal, szintén tetszőleges mennyiségekkel.

Most már lefuttathatjuk az ARUFESU összefésülő programot, amely létrehozza az összefésült állományt. Az eredmény a képernyőn látható:

```
 OSSZEFESULES
 ------------
 EGYIK+MASIK ==> OSSZEFESULT
1111
3333
4444
5555
6666
8888
KESZ
```

Természetesen az állományt magát is megnézhetjük, akár az adatonként olvasó programmal, akár (ARUTORZS-re való átnevezés után) az áruforgalmi listát készítő ARUGYJT programmal. (Ha az utóbbit választjuk, célszerű a mennyiségeket nem tetszés szerint választani, hanem úgy, hogy az eredmény könnyen ellenőrizhető legyen.)
A program egyébként jó példa a programstrukturálás lehetőségeinek változatosságára. Vessük össze a rendezett karbantartást végrehajtó RENKARB programmal! Ott a program funkcionális modulokra bomlott, vagyis minden alárendelt modul, hívott szubrutin meghatározott adatfeldolgozási funkcióval rendelkezett: így volt módosító, beszúró stb. modul. Az összefésülő programban ezek a funkciók a vezérlő modulban szerepelnek. A hívott szubrutinok nem adatfeldolgozási funkciót hajtanak végre, hanem egy-egy meghatározott gépi tevékenységet: adatátvitelt egy állományból, illetve egy állományba, azaz lemezről olvasást, vagy lemezre írást.
Végezetül felhívjuk a figyelmet arra, hogy ebben, valamint a megelőző fejezetben korántsem merítettük ki a soros állományok kezelésének összes lehetőségét. Csak néhány szemelvényt mutattunk be a kezelés legfontosabb alapelveinek illusztrálására. A közölt programok pusztán az igazi adatfeldolgozás lényegi tulajdonságainak kipróbálására alkalmas minták.

Adatfeldolgozó rendszerek

Könyvünk célja a soros adatállományokkal kapcsolatos adatfeldolgozási tevékenységek megismertetése, függetlenül attól, hogy azok a COMMODORE gépen megvalósíthatók-e, avagy sem, illetve hogyha igen, akkor a megoldás gazdaságos-e, avagy sem, ezért szólnunk kell az adatfeldolgozó rendszerekről is.
Amint láthattuk, a soros állományok kezelése általában nem oldható meg egyetlen programmal. Többnyire programok egész sorát kell megírnunk: külön felvivő, bővítő, lekérdező, feldolgozó, módosító stb. programokra van szükségünk. Kivéve persze, ha a gépünk elég nagy ahhoz, hogy mindezek a funkciók egyetlen óriási programba legyenek összefoghatók, annak funkcionális moduljait képezve.
A COMMODORE persze nem ilyen, de hát nem is adatfeldolgozásra készült. Arra viszont nagyon is alkalmas, hogy szinte bármilyen bonyolultságú adatfeldolgozó rendszert megfelelően lekicsinyítve kipróbálhassunk, sőt akár szimulálhassunk rajta.
Egyébként a COMMODORE-nak, mint fejlesztőgépnek a jelentősége éppen emiatt a szimulálási lehetőség miatt jóval nagyobb, mint gondolnánk. Ugyanis a programszerkezetet könnyebben lehet átvinni egyik gépről a másikra, mint a kódolt programot. Ha tehát megtervezzük egy adatfeldolgozó program szerkezetét, majd ennek COMMODORE-ra kódolt programját leteszteljük, akkor a szerkezet jóságát bizonyítottuk. Ha e program nem futtatható másik gépen, és ezért újra kell írnunk, ezt a már kipróbált szerkezet alapján tesszük, vagyis a program elvileg biztosan jó lesz — legfeljebb kódolási hibák lehetnek benne, azok javítása pedig egyszerűbb és gyorsabb, mint a konstrukciós hibáké.
Még hatékonyabban támaszkodhatunk a COMMODORE-ra a programtervezésben, ha rendelkezünk olyan szoftverrel, amely programszerkezetek formai ellenőrzésére, tesztelésére és dokumentálására alkalmas, ezzel hatékonyan támogatva a tervezést.
Márpedig ilyen szoftver Magyarországon létezik, és kapható is. Sőt, az említetteken kívül még arra is alkalmas, hogy egy előre elkészített programkönyvtárban tárolt modulokból, programrészletekből a megadott és a rendszer által ellenőrzött programszerkezet alapján összeállítsa a működőképes, futtatható programot, mégpedig úgy, hogy a szerkezeti elemek, például alternatívák, ciklusok stb. működéséhez, sőt az egymáshoz kapcsolódásukhoz szükséges utasításokat, hívásokat, vezérlésátadásokat automatikusan befordítja a programba, így ezeket soha nem kell magunknak kódolnunk. Minthogy magából a prog-

ramszerkezetből képezi ezeket a szoftver, soha nem lehetnek ellentmondásban a szerkezettel, és természetesen nem lehet bennük hiba.

Egy ilyen szoftver segítségével az általunk megtervezett programszerkezetet anélkül tesztelhetjük és dokumentálhatjuk, hogy magát a programot meg kellene írnunk. Ha az így szimulált program nem úgy működik, ahogyan elvárjuk tőle, akkor ennek csakis az lehet az oka, hogy a szerkezete nem jó — vagy azért mert rosszul terveztük meg, vagy azért, mert a jól megtervezett szerkezetet rosszul valósítottuk meg.

A szóban forgó szoftver további előnye, hogy a könyvünkben is bemutatott, egyszerű és jól áttekinthető szerkezeti elemekkel dolgozik, ami azt jelenti, hogy a tesztelendő programszerkezetet ilyen elemekből állíthatjuk össze, és a tesztelt programszerkezetet is ilyen ábrákkal dokumentálja.

KERETRENDSZER

Maradjunk tehát annál a megoldásnál, hogy egy soros állomány feldolgozásához több programot kell írnunk. Ezek kezelését nemcsak megkönnyíti, hanem biztonságosabbá is teszi, ha egységes programrendszerré fogjuk őket össze. Ez különösen érvényes a mikrogépes környezetben, ahol az üzemeltetést nem erre a célra külön kiképzett, hivatásos számítástechnikai szakemberek, operátorok végzik.

Példaként képezzünk programrendszert a költségviselők nyilvántartására írt programjainkból.

Ehhez mindenekelőtt írnunk kell egy vezérlő programot, amely kívánságunk szerint automatikusan betölti és végrehajtja a rendszer általunk kiválasztott programját, és annak végrehajtása után átveszi a vezérlést, és lehetőséget nyújt újabb adatfeldolgozási funkció kiválasztására mindaddig, amíg a programot le nem állítjuk (29. ábra).

A feltételek a következők:

F1: ha a kiválasztott funkció nem = "FELVITEL"
F2: ha a kiválasztott funkció nem = "KINYOMTATAS"
F3: ha a kiválasztott funkció nem = "LEKERDEZES"
F4: ha a kiválasztott funkció nem = "KARBANTARTAS"
F5: ha a kiválasztott funkció nem = "TORLES"
F6: ha a kiválasztott funkció nem = "ATNEVEZES"
F7: ha a kiválasztott funkció = "ALLJ"

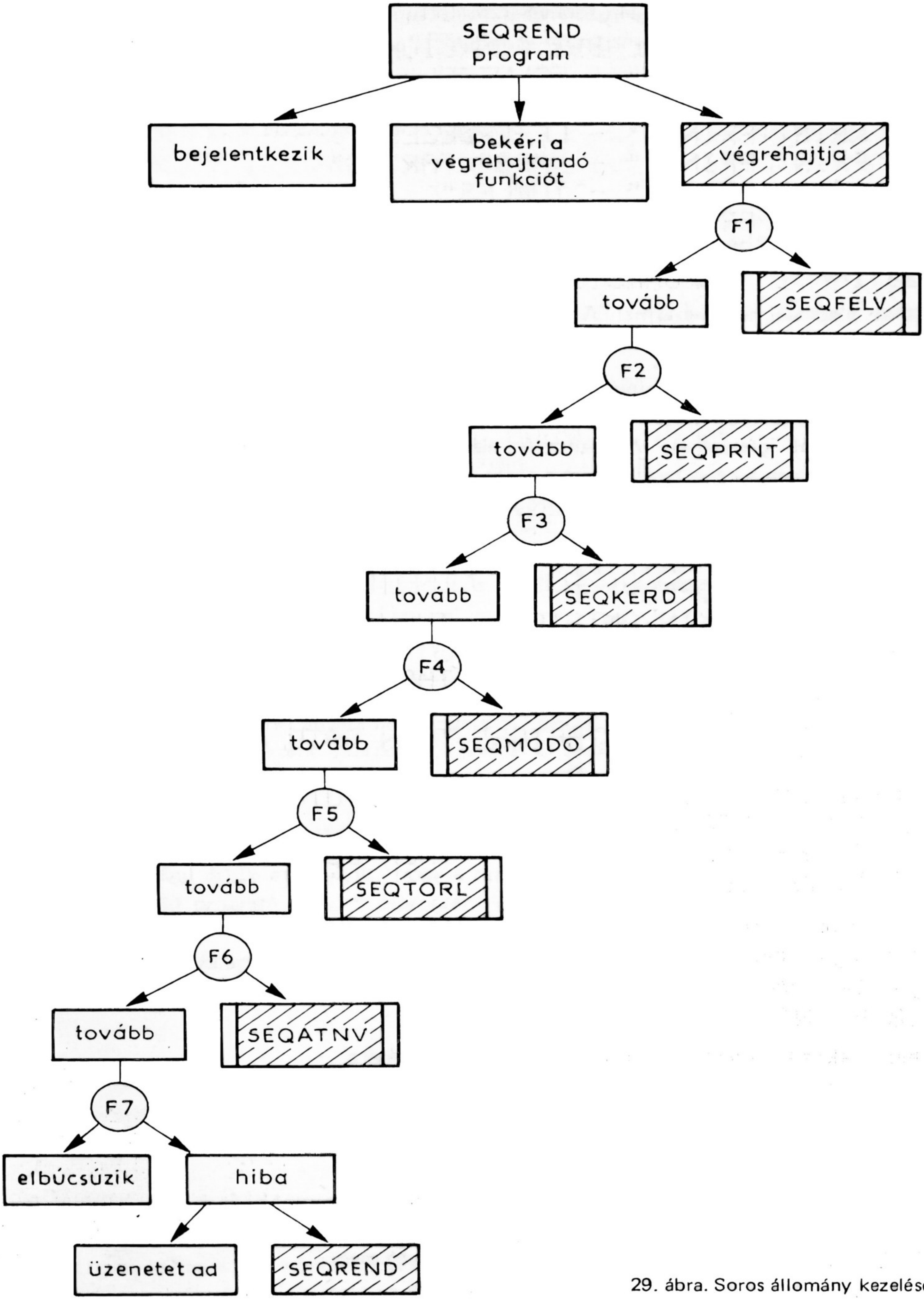
FIGYELEM!

A felvitt funkcióba a SEQFELV programnak az a módosított változata tartozik, amelybe már be van építve a bővítési lehetőség is. A karbantartási funkción a SEQMODO program tevékenységét értjük, amely tehát módosítani és törölni is tud.

NEW után a vezérlő programot is begépelhetjük.

A program szokás szerint bejelentkezéssel kezdődik:

```
1 PRINT "□"
2 PRINT
3 PRINT " KOLTSEGVISELOK NYILVANTARTASA"
4 PRINT " -----------------------------"
5 PRINT
```



29. ábra. Soros állomány kezelése

Majd feltálal egy menüt, azaz közli a választható funkciókat:

```
110 PRINT " AZ ALABBI FUNKCIOK VALASZTHATOK"
130 PRINT:PRINT " - FELVITEL"
131 PRINT:PRINT " - KINYOMTATAS"
132 PRINT:PRINT " - LEKERDEZES"
133 PRINT:PRINT " - KARBANTARTAS"
134 PRINT:PRINT " - TORLES"
135 PRINT:PRINT " - ATNEVEZES"
140 PRINT
150 INPUT " █FUNKCIO█= ALLJ████████";F$
```

Itt az alapértelmezés szerint "ALLJ" választ a felsorolt funkciók valamelyikével kell felülírnunk.

Megjegyezzük, hogy a menüválasztásnak a bemutatott változata eléggé primitív. Ennél lényegesen jobb menükezelési lehetőségek is léteznek, amelyek sokkal közelebb állnak a felhasználók igényeihez. Mi csak azért elégedtünk meg ezzel a módszerrel, mert e kötetünk célja nem a képernyő, hanem a soros állományok kezelésének a bemutatása.
A kiválasztott funkciót végrehajtó programot a vezérlő program automatikusan betölti, és azonnal el is indítja:

```
210 IF F$<>"FELVITEL"     THEN GOTO 310
220 LOAD "SEQFELV",8
310 IF F$<>"KINYOMTATAS"  THEN GOTO 410
320 LOAD "SEQPRNT",8
410 IF F$<>"LEKERDEZES"   THEN GOTO 510
420 LOAD "SEQKERD",8
510 IF F$<>"KARBANTARTAS" THEN GOTO 610
520 LOAD "SEQMODO",8
610 IF F$<>"TORLES"       THEN GOTO 710
620 LOAD "SEQTORL",8
710 IF F$<>"ATNEVEZES"    THEN GOTO 810
720 LOAD "SEQATNV",8
```

Ha nemlétező funkciót választottunk, például hibásan gépeltük be a funkció megnevezését, a program hibaüzenetet ad:

```
810 IF F$="ALLJ"          THEN GOTO 910
820 PRINT
```

Majd várakozik, hogy legyen időnk az üzenet elolvasására:

```
830 PRINT " ";F$;" █NEM VALASZTHATO!█"
840 FOR I=1 TO 1800
850 NEXT I
```

A hibás válaszból nem tudja eldönteni, hogy melyik programot hívja meg, ezért saját magát fogja meghívni. A vezérlő program viszont most eleve bent van a tárban, ezért a hívás betöltés nélkül, egyszerű vezérlésátadással valósítható meg:

```
860 GOTO 1
```

Ha viszont a programot leállítottuk, azaz nem adtunk meg funkciót, vagyis az "ALLJ" szöveg meghagyásával nyomtuk le a RETURN gombot, akkor a program elbúcsúzik, és leáll:

```
910 PRINT
920 PRINT " VISZONTLATASRA!"
999 END
```

FIGYELEM!
Az így működő programrendszerben a COMMODORE esetén a vezérlő programnak kell elfoglalnia a legnagyobb tárterületet.

Ezért a vezérlő programot ki kell egészítenünk tartalmilag érdektelen sorokkal, célszerűen megjegyzésekkel, hogy a mérete esetünkben legalább 7 blokknyi legyen:

```
2000 REM: ----------------------------------------
2001 REM: A PROGRAMTERULET MEGNOVELESE
2002 REM: ABBOL A CELBOL, HOGY AZ ELSO
2003 REM: BETOLTOTT PROGRAM, (A  KERET
2004 REM: RENDSZER) FOGLALJA EL A LEG-
2005 REM: NAGYOBB TARTERULETET.
2010 REM: ----------------------------------------
2011 REM: A PROGRAMTERULET MEGNOVELESE
```

és hasonló módon így tovább.

```
2045 REM: NAGYOBB TARTERULETET.
2050 REM: ----------------------------------------
```

Most mentsük ki a vezérlő programot SEQREND néven, de ne indítsuk el!!!
Ahhoz, hogy a rendszerünk működjön, feltétlenül szükséges, hogy a vezérlő SEQREND program kivételével, az általa hívott összes többi SEQ programban kicseréljük a 999 END utasítást:

```
999 LOAD "SEQREND",8
```

utasításra. Ezzel érjük el ugyanis, hogy a rendszerünk automatikusan működjön; vagyis hogy a vezérlő program minden alprogram végrehajtódása után újra betöltődjön, és átvegye a vezérlést.
A vezérlés átvétele ugyanis azt jelenti, hogy a program a betöltése után azonnal el is indul, mintha RUN parancsot kapott volna. Ez a programból kiadott LOAD hatására következik be.

FIGYELEM!
Tehát míg a parancsként kiadott LOAD után RUN parancsot is ki kell adnunk a program elindításához, az utasításként kiadott LOAD után ezt nem kell, sőt *nem is szabad megtennünk!*

Megjegyezzük, hogy az END utasítást eddig csak azért használtuk, hogy a programunk kipróbálásával ne kelljen megvárnunk a vezérlő program elkészültét.
Ami pedig a vezérlő program REM-ekkel való feltöltését illeti, annak az oka, hogy egy programból kiadott LOAD utasítással csak a hívó programnál kisebb programot lehet be-

hívni. Ez a programterületnek és adatterületnek COMMODORE-nál alkalmazott kezelési módjából adódik.
Ez a probléma nemcsak a fentebb ismertetett primitív és egyáltalán nem elegáns módon hidalható át, de a bemutatott példa minden körülmények között használható, és nem kell hozzá gépi szinten ismernünk a COMMODORE működését, tárkezelési és programkezelési módját.
A más módok ismertetése meghaladja sorozatunk e kötetének a feladatát, így most a számos egyéb lehetőség közül csak egyet említünk meg. Ha a programjainkban levő összes LOAD helyére a

```
POKE 45,120:POKE 46,120:CLR:LOAD "PROGRAMNEV",8
```

sort kódoljuk, akkor a COMMODORE úgy fogja tárolni az adatokat, hogy a programbetöltések azokat nem fogják zavarni – és persze ilyenkor nincs szükségünk a vezérlő program végén levő helykitöltő megjegyzésekre sem.
Sorozatunk egy későbbi kötetében erre a kérdésre még visszatérünk, és részletesen tárgyalni fogjuk a jelenség okait, kivédésének lehetőségeit. Ennek kapcsán fogjuk elmagyarázni a fenti, jelenleg még rejtélyes utasítások szerepét és hatását.
Most tehát ott tartunk, hogy a rendszerhez tartozó összes programunkba bevezettük a jelzett módosításokat. Ekkor betölthetjük a SEQREND programot, és elindíthatjuk.

```
KOLTSEGVISELOK NYILVANTARTASA
-----------------------------
AZ ALABBI FUNKCIOK VALASZTHATOK
- FELVITEL
- KINYOMTATAS
- LEKERDEZES
- KARBANTARTAS
- TORLES
- ATNEVEZES
[illegible]=? KINYOMTATAS

KOLTSEGVISELOK NYILVANTARTASA
-----------------------------
AZ ALABBI FUNKCIOK VALASZTHATOK
- FELVITEL
- KINYOMTATAS
- LEKERDEZES
- KARBANTARTAS
- TORLES
- ATNEVEZES
[illegible]=? NYOMTATAS
NYOMTATAS [illegible]
```

```
KOLTSEGVISELOK NYILVANTARTASA
-----------------------------
AZ ALABBI FUNKCIOK VALASZTHATOK
- FELVITEL
- KINYOMTATAS
- LEKERDEZES
- KARBANTARTAS
- TORLES
- ATNEVEZES
[illegible]=? ALLJ
VISZONTLATASRA!
```

A rendszert így próbálhatjuk ki a legegyszerűbben:

— FELVITEL: létrehozzuk a KOLTSEGVISELOK állományt.
— LEKERDEZES: lekérdezzük a KOLTSEGVISELOK állományt, mind teljes körű, mind egyedi lekérdezéssel.
— FELVITEL: bővítjük a KOLTSEGVISELOK állományt.
— KARBANTARTAS: szűkítjük és egyben módosítjuk is a KOLTSEGVISELOK állományt.
— TORLES: töröljük a KOLTSEGVISELOK állományt.
— ATNEVEZES: átnevezzük a MODOSITOTT állományt KOLTSEGVISELOK-re.
— KINYOMTATAS: kinyomtatjuk a KOLTSEGVISELOK állományt.
— ALLJ: leállítjuk a rendszert.

Mindez persze csak akkor lehetséges, ha az egyik fenti állományunk sincs a rendszer indításakor a lemezen. Ellenkező esetben, ha mégis a fenti lépésekkel akarjuk végigpróbálni a rendszert, akkor a meglevő állományok törlésével kell kezdenünk a kipróbálást.

Az ilyen rendszert, amely tehát önmaga nem hajt végre semmilyen adatfeldolgozási funkciót, csupán egységes keretbe foglalja az egyes funkciókat végrehajtó különböző programokat, *keretrendszernek* nevezzük.

FIGYELEM!

Ez a rendszer nem üzemi, hanem mintarendszer. Csak egy illusztráció a hasonló jellegű rendszerek tervezéséhez.

Egy kivétellel nincs például a programokba beépítve a hibavédelem. Vagyis az, hogy mi legyen, ha nem találja a feldolgozandó állományt, vagy a létrehozandó állomány már szerepel a lemezen.

Ez a parancs-csatorna lekérdezésével könnyen megoldható, mint ahogyan azt a SEQTORL törlőprogramban is tettük.

Csak azok a programok szerepelnek a rendszerben, amelyek a soros állományok kezeléséhez elengedhetetlenül szükségesek. Természetesen a rendszert tetszés szerint módosíthatjuk, illetve bővíthetjük. Például felvehetjük külön funkcióként a válogatási lehetőséget; szétválaszthatjuk a felvitelt létrehozásra és bővítésre; ugyanígy a karbantartást törlésre és

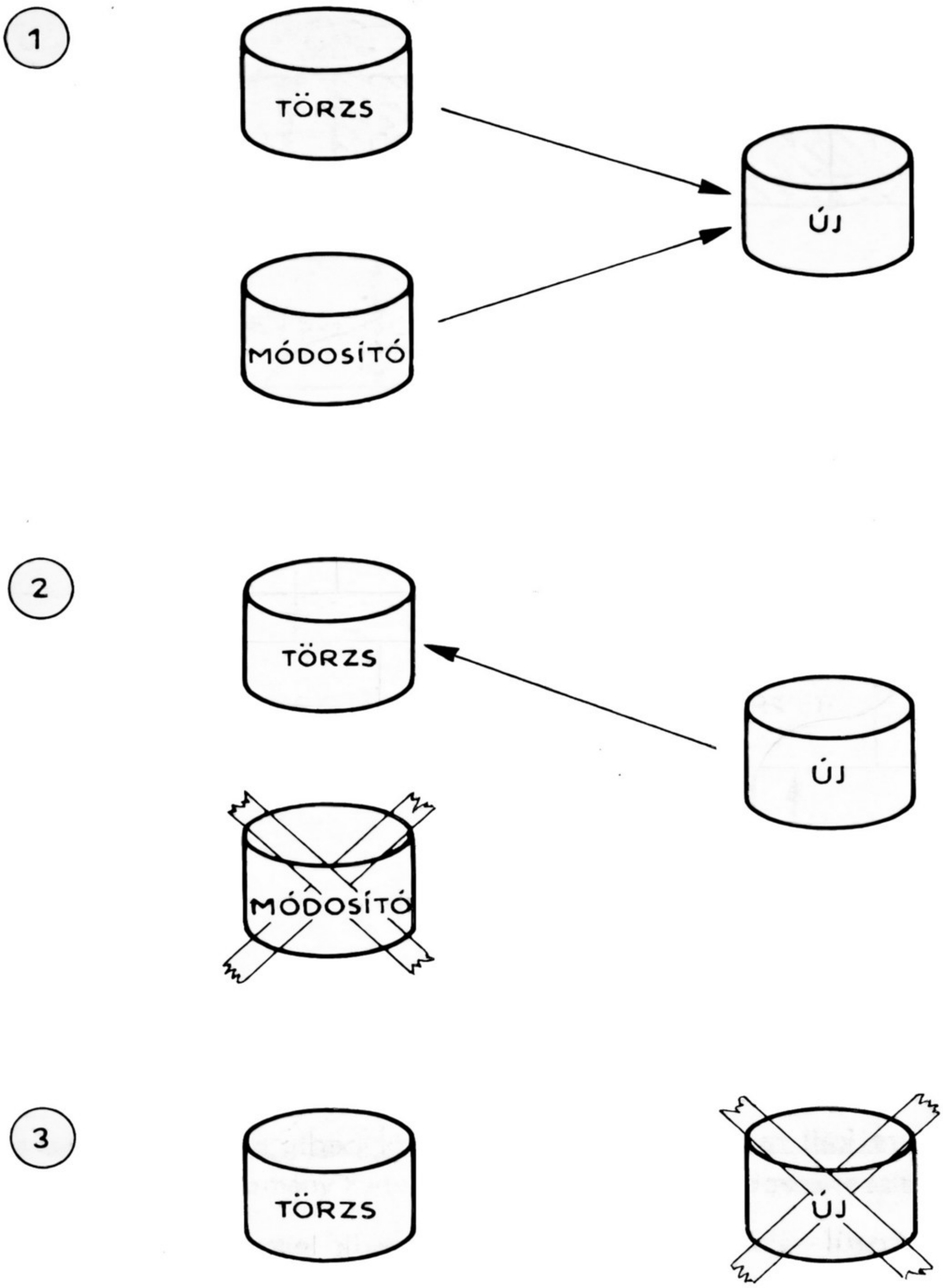
módosításra; kibővíthetjük a másolással és így tovább. Akár kötetünk későbbi programjaiból is válogathatunk.
Gyakorlásképpen készítsünk hasonló rendszert az ARUTORZS kezeléséhez is. (A megoldást nem közöljük — tekintsük házi feladatnak.)

RENDEZETLEN KARBANTARTÁS*

A soros állományokkal kapcsolatos legfontosabb és legbonyolultabb funkciók egyike az állomány adatainak a módosítása.

A soros állomány adatatait ugyanis nem lehet felülírni. Ez ellen egyébként a lemezkezelő külön is védekezik: az írásra vagy olvasásra megnyitott állomány állapota feldolgozás közben nem változtatható meg.
Ha például egy költségviselő adatai megváltoznak, akkor nem tehetjük meg, hogy lekérdezzük a rekordját, majd a beolvasott adatokat módosítjuk, és a módosított rekordot visszaírjuk az állományba. (Mint majd látjuk, relatív vagy random állomány esetén ezt meg lehet csinálni, de a sorosnál nem.)
A módosítás tehát csak úgy valósítható meg, hogy egy új állományt hozunk létre.
A már ismert példában (lásd SEQMODO!) ezt így is tettük. Ott a módosítást párbeszédes módon oldottuk meg, aminek sikeressége azonban függ a gépkezelő képességeitől, felkészültségétől, pillanatnyi teljesítményétől.
A rendezett állományoknál (lásd: RENKARB!) másfajta karbantartást láttunk: a törzsállományt egy módosító állomány adataival a program automatikusan, operátori beavatkozás nélkül módosította.
Kérdés, hogy ezt a rendezetlen állományokkal meg lehet-e csinálni? Hiszen a módszernek vannak szemmel látható előnyei: hibák csak az adatfelvitelnél fordulhatnak elő. A válasz lehangoló. Ugyanis ilyenfajta karbantartás elvileg megoldható, de minthogy körülményes és lassú, gyakorlatilag nem jöhet számításba.
Ennek ellenére a megoldást bemutatjuk, egyrészt azért, hogy az olvasó is lássa, mire alapozzuk a fenti kijelentést; másrészt azért, mert tanulságos; harmadrészt pedig a soros állományok kezelésének megértéséhez közelebb visz egy lépéssel.
Induljunk ki tehát a költségviselők nyilvántartásából, és vegyük a legegyszerűbb esetet, amikor nem engedjük meg a rekordokon belüli egyes adatok önálló módosítását, hanem mindig a teljes rekordot kell átírnunk, ha annak csak egyetlen adata is változott. (Ezt nevezzük rekordonkénti módosításnak, szemben a már bemutatott, tételenkénti módosítással.) Ilyenkor tehát nem módosító adataink, hanem módosító rekordjaink vannak.
Ezeket egy módosító állományba visszük fel, miáltal az adatfelvitel és a módosítás végrehajtása szétválasztható (30. ábra).
A rekordonkénti módosítás lényege, hogy vesszük az eredeti, a módosítandó állományunkat, amelyet törzsállománynak is nevezünk, valamint a módosító rekordokat tartalmazó módosító állományt, és ezek rekordjaiból összeállítjuk az új, a módosított állományt.
Ilyenkor persze gondoskodnunk kell arról, hogy a keletkezett új állomány legyen a továbbiakban a törzs. Ezután a módosító állományra már nincs szükségünk.
Bonyolítja a helyzetet, hogy nemcsak a létező rekordokban lehet adatváltozás, hanem egyes költségviselők megszűnhetnek, vagy újak keletkezhetnek. Ilyenkor meglevő rekor-

*E módszert csak a belőle levonható tanulságok kedvéért mutatjuk be. Közvetlen gyakorlati felhasználásra nem javasoljuk.

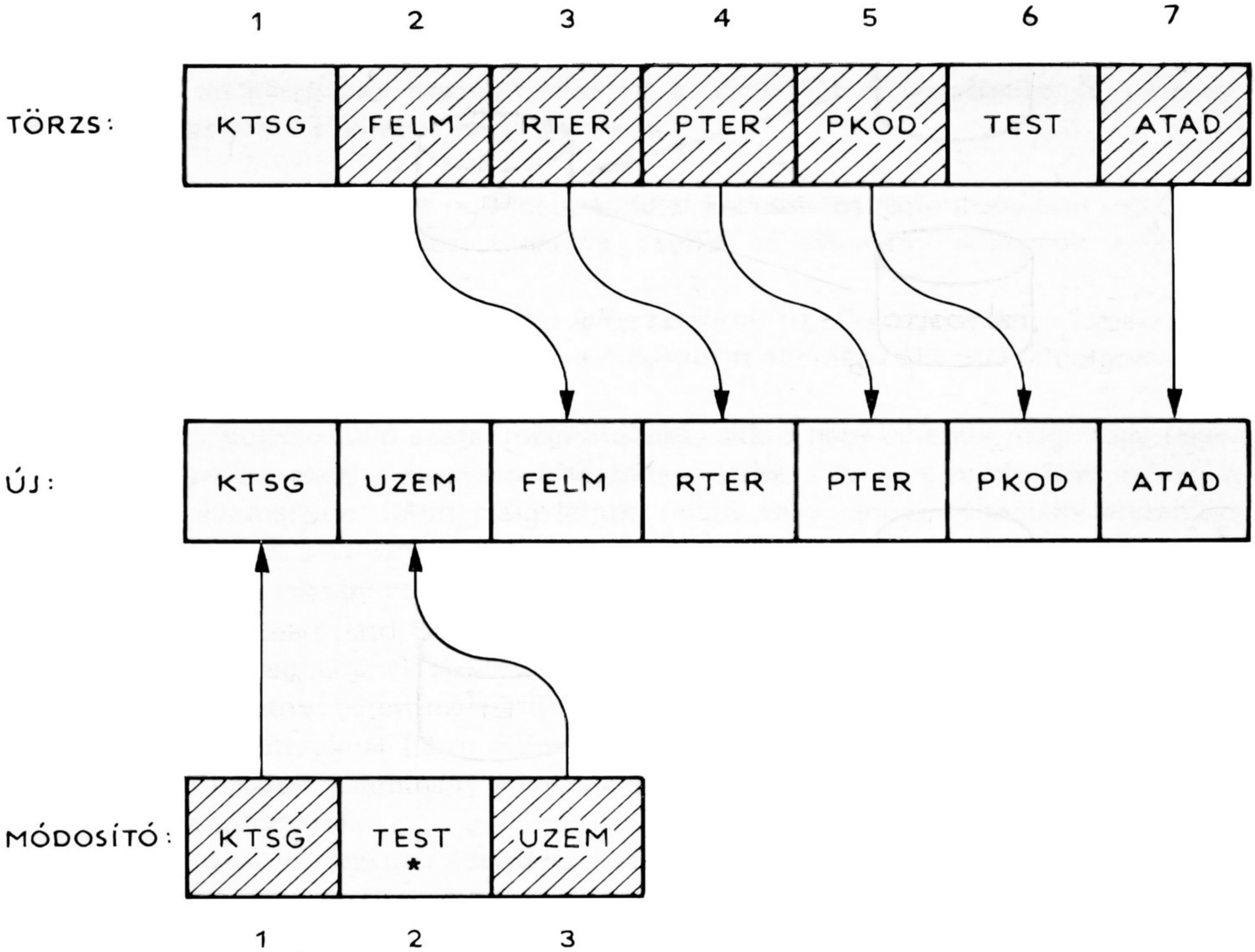


30. ábra. A soros karbantartás menete

dokat kell törölnünk, illetve új rekordokat kell felvinnünk. Tehát nemcsak a rekordok tartalma változhat, hanem az állomány terjedelme is (31. ábra).

Mint már a rendezett karbantartásnál is láttuk, három fő funkciót kell végrehajtanunk:
- BESZÚRÁS: ha egy módosító rekordnak nincs törzsbéli párja, és a módosítás nem törlés. (UZEM)
- VÁLTOZATLANUL HAGYÁS: ha egy törzsrekordnak nincs módosító párja. (FELM)
- MÓDOSÍTÁS: ha egy törzsrekordnak van módosító párja. Ez két további funkcióra bomlik:
- CSERE: ha a módosítás nem törlés. (KTSG)
- TÖRLÉS: ha a módosítás törlés. (TEST)

A karbantartás tehát itt is a párosításra épül. Ehhez pedig szükségünk van a keresésre. De most nem használhatjuk a rendezett karbantartásnál megismert keresést, hanem a lekérde-



31. ábra. Soros állomány karbantartásának elve

zésnél megismert soros keresésre kell támaszkodnunk. Ez a tény pedig a már megismerttől teljesen eltérő karbantartási szerkezethez vezet.

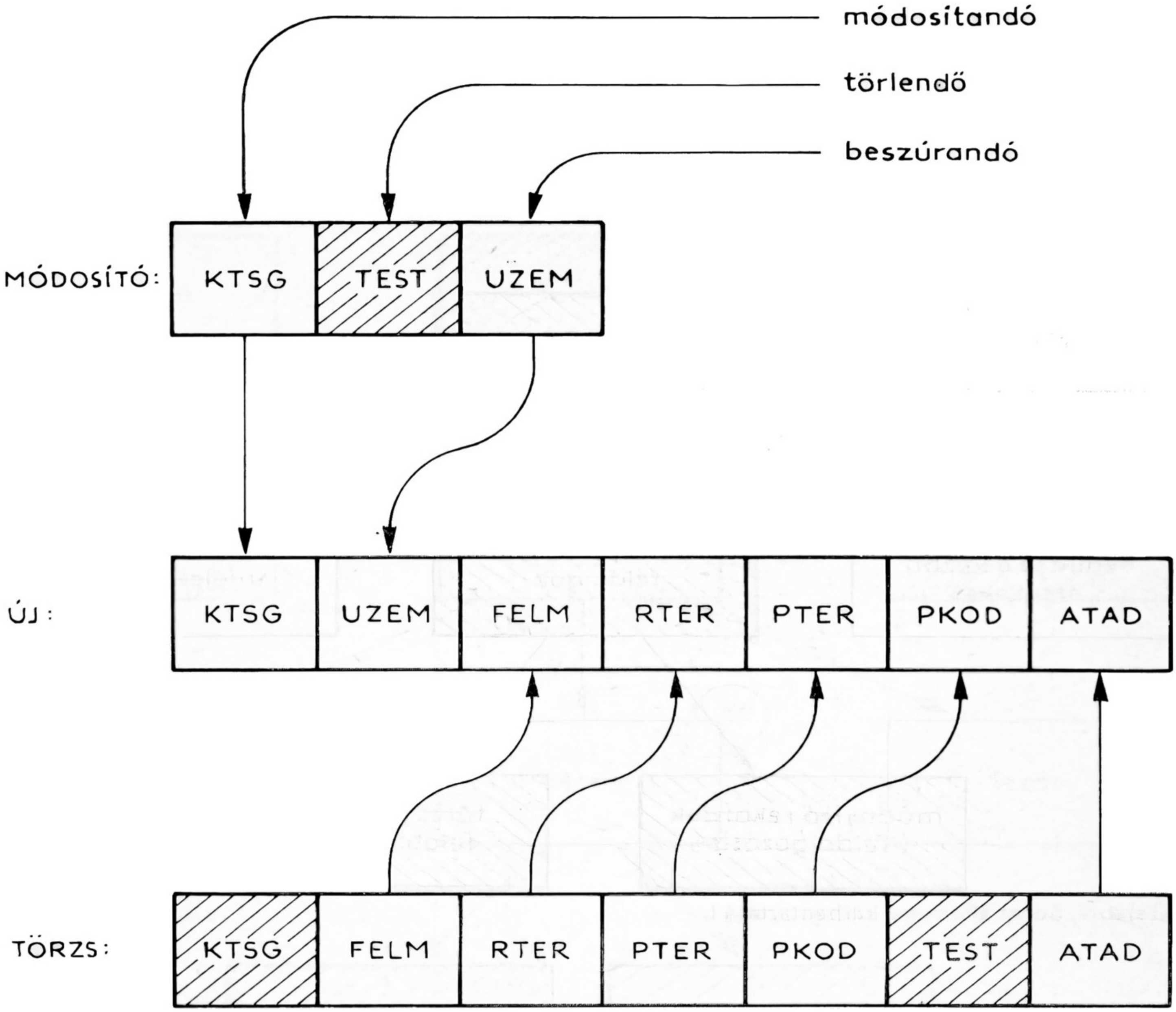
Minthogy többféle megoldás közül választhatunk, vegyük az egyik legegyszerűbbet. Ennek az a lényege, hogy az új állományba csak a módosított, beszúrandó és a változatlanul hagyott rekordokat visszük át. Az előbbieket a módosító állományból, az utóbbit pedig a törzsből. Vagyis a módosítást két lépésben hajtjuk végre: először átvisszük a módosító állományból az újba mindazokat a rekordokat, amelyek nem egy törzsrekord törlését határozzák meg; majd a törzsből átviszünk minden olyan rekordot, amelynek a módosító állományban nincs párja (32. ábra).

Így minden egyes törzsrekordhoz végig kell keresnünk a módosító állományt, hogy a párt megtaláljuk, vagy hogy meggyőződjünk arról, hogy nincs. A 33.a) ábrán látható karbantartás feltétele

F1: ha mehet = "I"

Ezen belül a módosító rekordok feldolgozásához a 33.b) ábra szerint szükséges feltételek

F2: ha van még módosító rekord, azaz K$ nem = VEGE$
F3: ha a rekord nem törlendő, azaz M$ nem = "TORLENDO"



32. ábra. Soros állomány karbantartásának egy lehetséges megvalósítása

A törzsrekordok feldolgozásának a 33.c) ábra szerinti feltételei pedig:

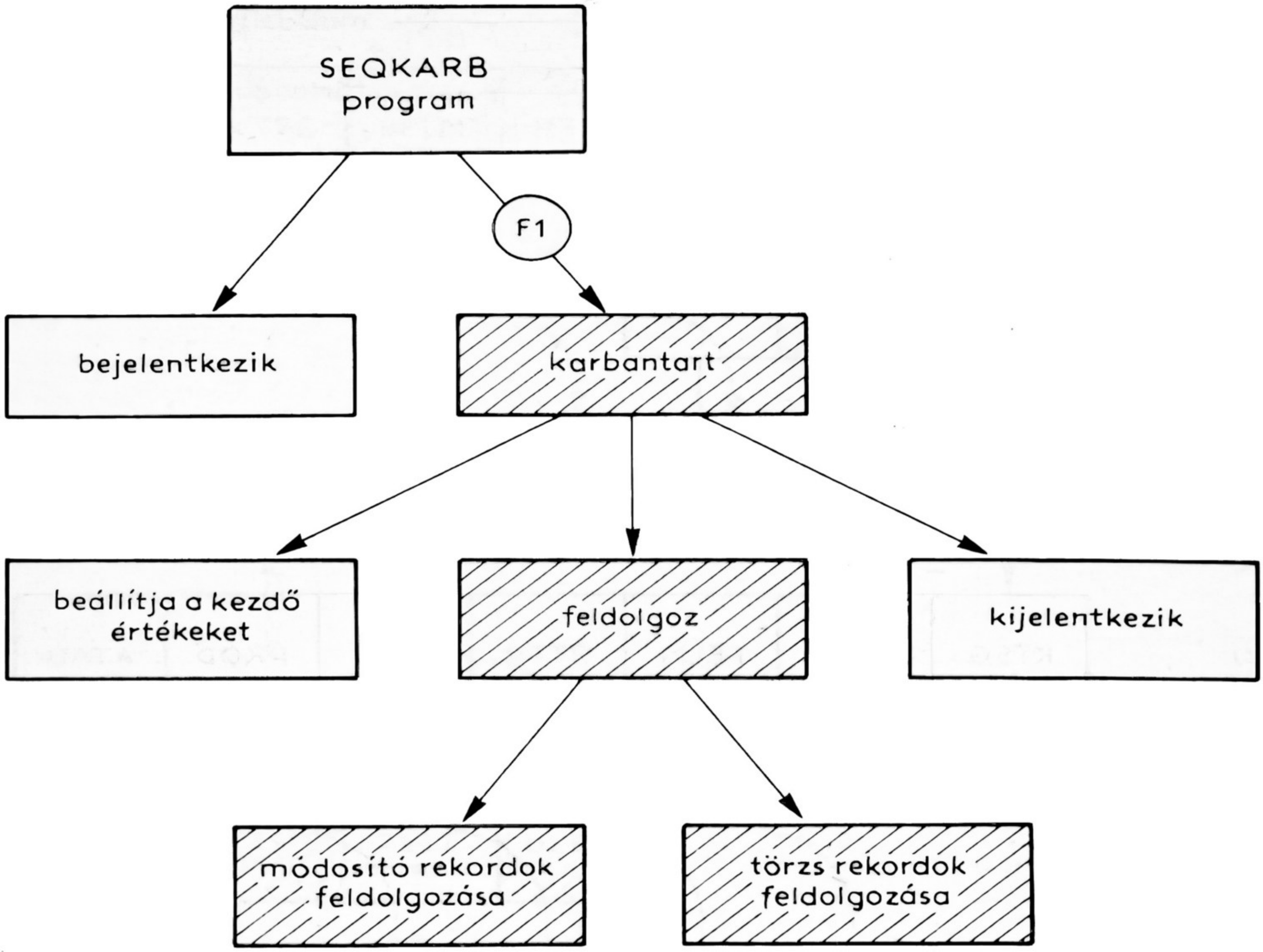
F4: ha van még törzsrekord, azaz K$ nem = VEGE$

F5: ha van még módosító rekord, azaz KK$ nem = VEGE$, és annak KK$ kódja nem = a törzsrekord K$ kódjával

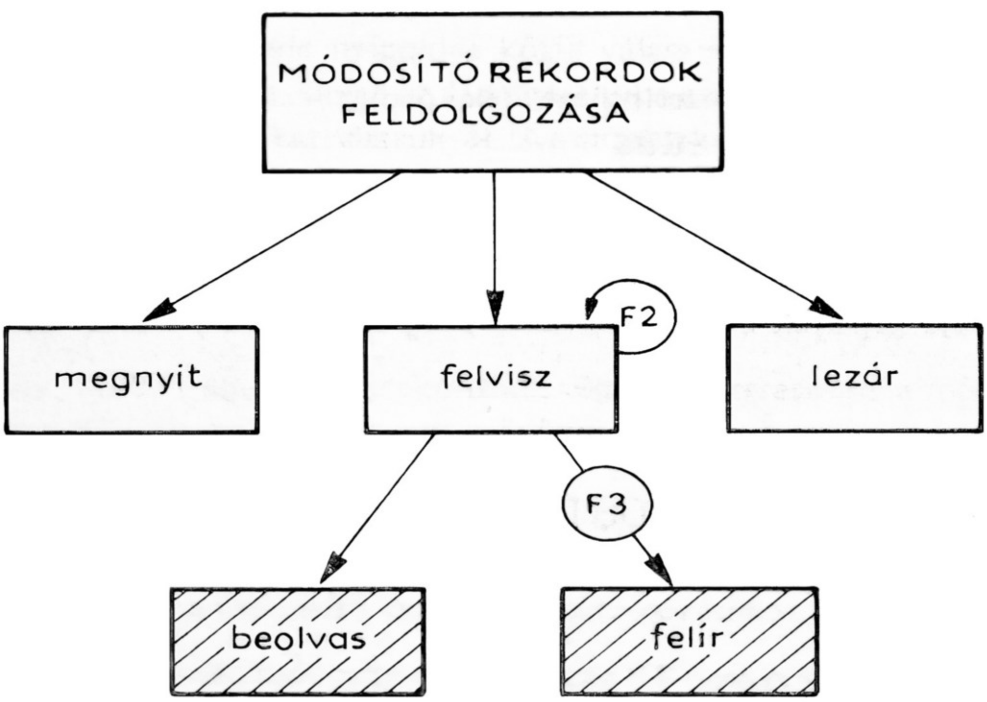
F6: ha nincs több módosító rekord, azaz KK$ = VEGE$

Adjunk ki NEW parancsot, majd gépeljük be a karbantartó programot, amely először bejelentkezik:

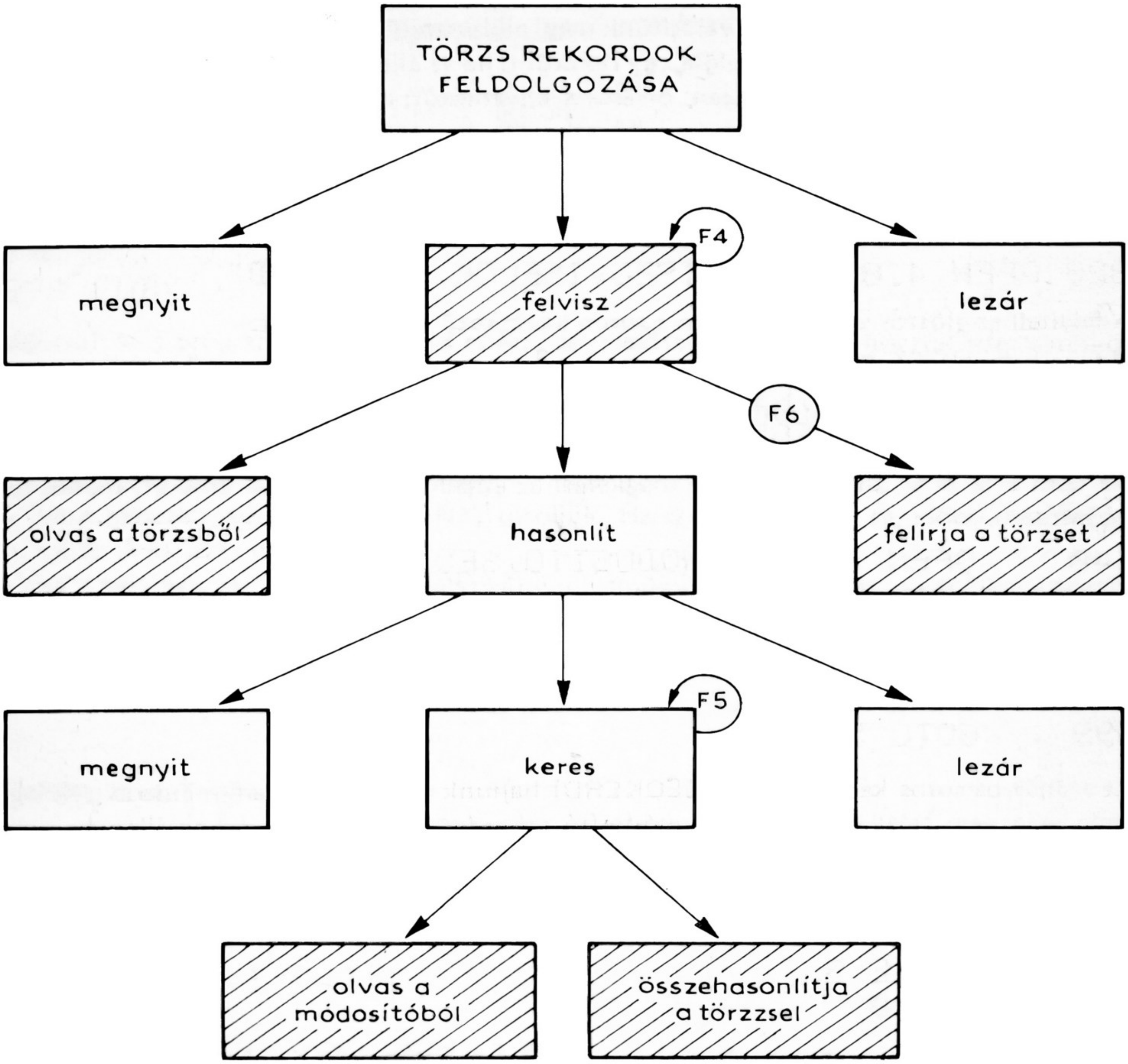
```
1 PRINT "□"
2 PRINT
3 PRINT " KOLTSEGVISELOK MODOSITASA"
4 PRINT " ___________________________"
5 PRINT
6 PRINT
7 INPUT " MEHET =I/N=";V$
8 IF V$<>"I" THEN GOTO 999
```



33.a) ábra. Soros állomány karbantartása I.



33.b) ábra. Soros állomány karbantartása II.



33.c) ábra. Soros állomány karbantartása III.

Majd beállítja a kezdőértékeket, azaz a VEGE$ végjelet és az S$ elhatároló jelet:

```
10 VEGE$="ππππ"
20 S$=CHR$(13)
```

Írásra megnyitja az új állományt UJALLOMANY néven:

```
110 OPEN 2,8,2,"UJALLOMANY,SEQ,WRITE"
```

Ezután következik a nem törlendő módosító rekordok felvitele az új állományba:

```
120 OPEN 3,8,3,"MODOSITO,SEQ,READ"
210 K$=VEGE$
220 :   INPUT#3,K$,M$,T,R
230 :   IF K$=VEGE$ THEN GOTO 310
240 :   IF M$="TORLENDO" THEN GOTO 299
250 :   PRINT#2,K$;S$;M$;S$;T;S$;R
299 GOTO 210
310 CLOSE 3
```

Itt egyszerű soros feldolgozást valósítunk meg, elöltesztelő ciklussal: a program megnyitja a módosító állományt; olvas belőle egy rekordot; ha az állománynak nincs vége, megnézi, hogy a rekord törlendő-e; ha igen, olvassa a következőt; ha nem, akkor felviszi az új állományba, majd ezután olvassa a következőt; ha a módosító állomány véget ért, lezárja.

Most következik a törzsrekordok felvitele az új állományba, ami a törzs megnyitásával kezdődik:

```
320 OPEN 4,8,4,"KOLTSEGVISELOK,SEQ,READ"
```

A felvitelhez először veszi a program a soron következő törzsrekordot:

```
410 K$=VEGE$
420 :   INPUT#4,K$,M$,T,R
430 :   IF K$=VEGE$ THEN GOTO 810
```

Ha nem érte el az állomány végét, megkeresi az éppen beolvasott törzsrekord módosító párját:

```
440 :   OPEN 3,8,3,"MODOSITO,SEQ,READ"
510 :   KK$=VEGE$
520 :   :   INPUT#3,KK$,MM$,TT,RR
530 :   :   IF KK$=VEGE$ THEN GOTO 710
540 :   :   IF KK$=K$    THEN GOTO 610
599 :   GOTO 510
```

Itt szabályos soros keresést (lásd: SEQKERD) hajtunk végre: a program mindaddig keres, amíg meg nem találja a megfelelő módosító rekordot, vagy el nem éri az állomány végét.

Ha megvan a törzsrekord módosító párja, a program nem visz fel az új állományba semmit, egyszerűen lezárja a módosító állományt, és veszi a következő törzsrekordot:

```
610 :   CLOSE 3
699 GOTO 410
```

Világos, hogy ilyenkor nem szabad egy rekordot sem felvinni az új állományba, hiszen ha a törzsnek van módosító párja, akkor a törzsrekord módosítandó vagy törlendő. Az utóbbi esetben egyáltalán nem szabad felvinni rekordot, az előbbiben pedig a módosítót kellene felvinni – azt viszont a program (a 120–310 sorokban) már felvitte.

Ha elértük a módosító állomány végét, ez azt jelenti, hogy az aktuális törzsrekordnak nem volt módosító párja. Ilyenkor a program lezárja a módosító állományt, felviszi a páratlan törzsrekordot az új állományba, majd áttér a következő törzsrekord feldolgozására:

```
710 :   CLOSE 3
720 :   PRINT#2,K$;S$;M$;S$;T;S$;R
799 GOTO 410
```

Ha a program a törzsállomány végére ért, lezárja mind az immár régi törzset, mind az új állományt:

```
810 CLOSE 4
820 CLOSE 2
```

Majd üzenetet ad a módosítás végrehajtásáról:

```
910 PRINT
920 PRINT " MODOSITAS KESZ"
930 PRINT
940 PRINT " ADATOK AZ UJALLOMANY-BAN"
950 PRINT
```

Végül leáll:

```
999 END
```

Mentsük ki a programot SEQKARB néven, de mielőtt elindítanánk, hozzuk létre a működéséhez szükséges környezetet:

- Ha a lemezen van KOLTSEGVISELOK állomány, és az még az eredetileg felvitt adatokat tartalmazza, nem kell tennünk vele semmit. Egyébként állítsuk elő a KOLTSEGVISELOK állományt az első felvitelkor megadott adatokkal.
- Ha a lemezen van UJALLOMANY, töröljük. Ha szükségünk van rá, előbb más néven mentsük ki, azaz másoljuk át.
- Ha a lemezen van MODOSITO állomány, ugyanúgy járjunk el, mint az új állománnyal. Majd hozzuk létre az alábbi MODOSITO állományt:

```
EZ FOG A LEMEZRE KERULNI:
-------------------------

[illegible] = KTSG
[illegible] = KOLTSEGHELYI ALTALANOS
[illegible] = 200
[illegible] = 100
[illegible] =I/N=? I

EZ FOG A LEMEZRE KERULNI:
-------------------------

[illegible] = TEST
[illegible] = TORLENDO
[illegible] = 0
[illegible] = 0
[illegible] =I/N=? I

EZ FOG A LEMEZRE KERULNI:
-------------------------

[illegible] = UZEM
[illegible] = BEUZEMELES
```

```
[illegible] = 80
[illegible] = 50
[illegible] =I/N=? I
```

Most már betölthetjük és elindíthatjuk a SEQKARB programot. Ha nem vétettünk hibát, akkor legfeljebb 1 percen belül el kell végeznie a karbantartást.
Az új állomány létrejöttéről a tartalomjegyzék lekérésével győződhetünk meg, a karbantartás sikerességéről pedig egy teljes körű lekérdezés (SEQKERD) végrehajtásával. A kinyomtatás (SEQPRNT) ehhez nem elég, mert azon a módosítás helyessége nem ellenőrizhető:

```
KOLTSEGVISELOK LISTAJA

KOD:   | MEGNEVEZES:
-------+--------------------------------

KTSG --- KOLTSEGHELYI ALTALANOS

UZEM --- BEUZEMELES

FELM --- HELYZETFELMERES

RTER --- RENDSZERTERVEZES

PTER --- PROGRAMTERVEZES

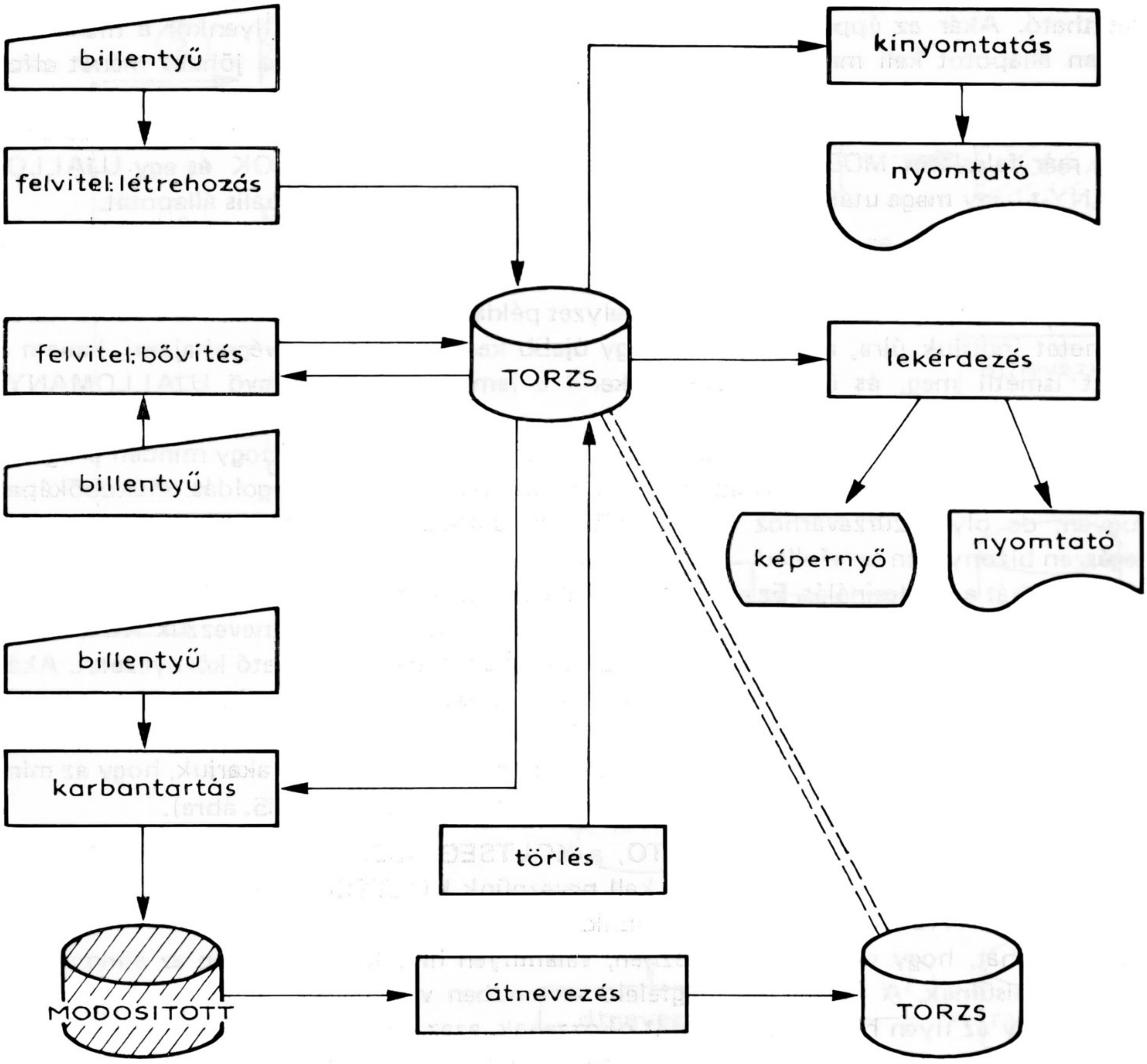
PKOD --- PROGRAMKODOLAS

ATAD --- RENDSZER ATADASA
```

Megjegyezzük, hogy a SEQKARB program a SEQREND rendszerbe, a SEQMODO program helyére beépíthető. Ennek csupán a könnyebb kipróbálás céljából van értelme.

FELDOLGOZÁSI MENETEK

A soros állományt általában lehetetlen egyetlen menetben feldolgozni. Láthattuk, hogy először szükség van egy *felvivő* mementre, amit egy ellenőrzésnek kell követnie. Ez történhet géppel, azaz ellenőrző programmal, vagy kinyomtatott lista alapján manuálisan. Így is, úgy is egy *ellenőrző* menetről van szó. Ennek következménye egy *bővítő* vagy *módosító* menet lehet, vagy esetleg mindkettő. Rendezett állományoknál ezek egyetlen *karbantartó* menetté olvadhatnak össze. Itt persze lehet külön *rendező* menet, vagy a módosító állományt létrehozó felvivő menet.
És ezzel még el sem kezdtük a tulajdonképpeni feldolgozást. Lekérdezésről, válogatásról például még szó sem volt (34. ábra).



34. ábra. Feldolgozási menetek

Különösen bonyolult a helyzet a karbantartó menet esetén, mert ez még önmagában is ismétlődhet. Nincs rá garancia, hogy egy sikeres karbantartás után nem merül fel újabb karbantartási igény. Sőt, általában az a normális állapot, hogy az állományt rendszeresen vagy alkalomszerűen, de többször és ismételten karban kell tartani. Ezekhez az üzemszerű karbantartásokhoz még jönnek azok, amelyeket a hibásan felvitt adatok javítására hajtunk végre.

Persze nemcsak a karbantartó menet lehet ilyen. Nyilvánvaló, hogy például a lekérdezés, válogatás, feldolgozás, kinyomtatás végrehajtására is lehet ismétlődő igény.

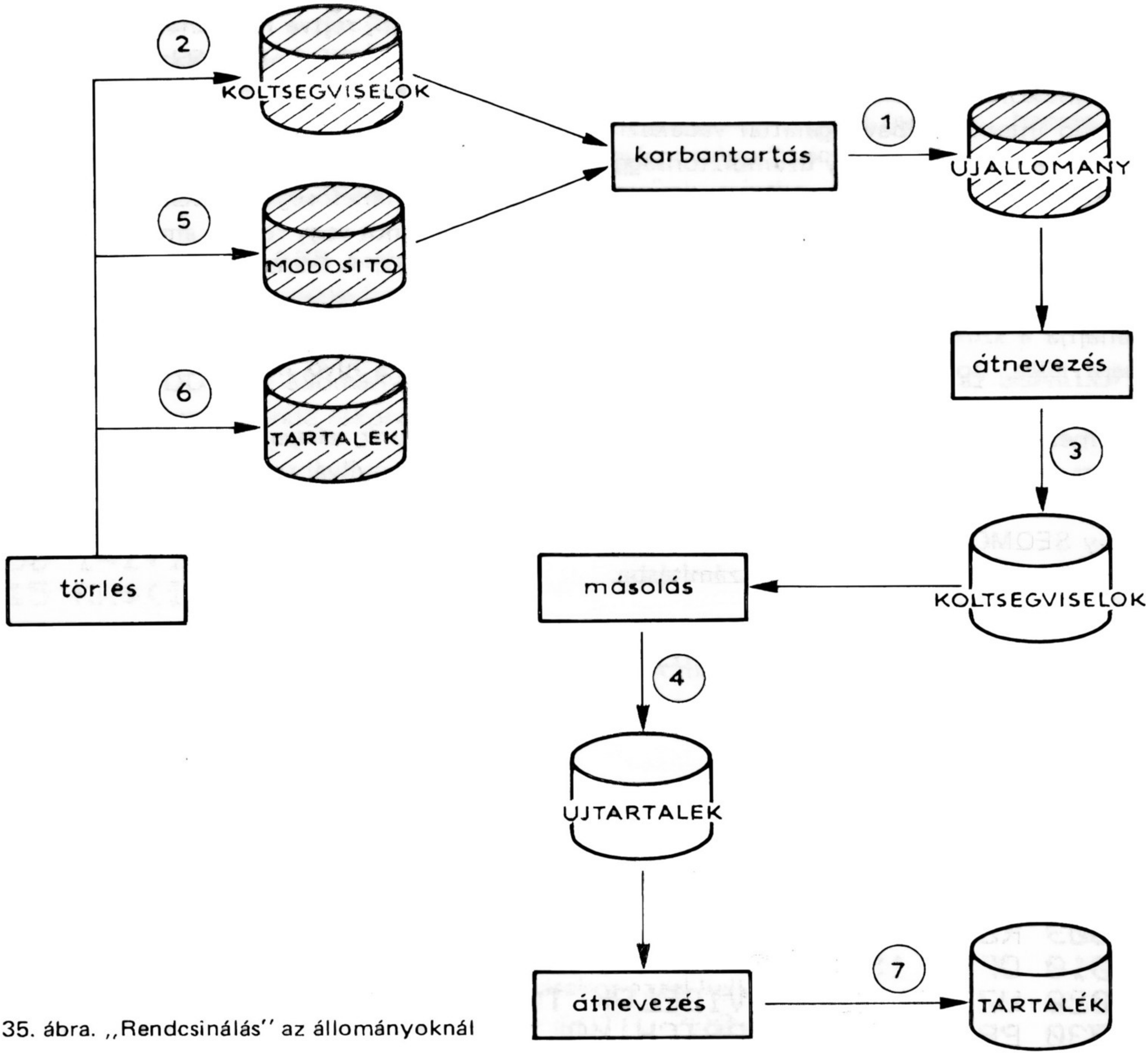
Szükségszerű tehát, hogy biztosítsuk a menetek megfelelő kapcsolatát. Ennek az a szokásos módja, hogy minden menet „rendet csinál" maga után. Ami azt jelenti, hogy olyan feldolgozási környezetet állít elő, amelyben a soron következő menet gondtalanul végrehajtódhat.

Persze gyakran előfordul, hogy egy adott feldolgozási menetet nem egy meghatározott másik menet követ, hanem a feldolgozási igénytől függően több különböző menet is vá-

lasztható. Akár az éppen befejeződött menet is megismételhető. Ilyenkor a menetnek olyan állapotot kell maga után hagynia, hogy azt bármelyik szóba jöhető menet elfogadhassa.
Tekintsük az imént bemutatott SEQKARB programot. Ez a sikeres végrehajtódása után egy már felesleges MODOSITO, egy már elavult KOLTSEGVISELOK és egy UJALLOMANY-t hagy maga után. Az utóbbi tartalmazza a nyilvántartás aktuális állapotát.

Nyilvánvaló, hogy ez a környezet a további menetek számára elfogadhatatlan. Ha lekérdezni akarunk, akkor a lekérdező menet a régi, módosítatlan KOLTSEGVISELOK állományból fog dolgozni. Ugyanez lesz a helyzet például a kinyomtatással. Ha a karbantartó menetet indítjuk újra, akkor az nem egy újabb karbantartást fog végrehajtani, hanem a régit ismétli meg, és összeütközésbe kerül a lemezen még meglevő UJALLOMANY-nyal.
Amint a törlés és átnevezés kapcsán már említettük, az a megoldás, hogy minden program kérje be a bemenő és kimenő állományok nevét, csak átmeneti megoldás. Működőképes ugyan, de olyan zűrzavarhoz vezet, amibe az adatfeldolgozó rendszerünk előbb-utóbb egészen bizonyosan belefullad.
Marad tehát a rendcsinálás. Ez a SEQKARB esetén azt jelenti, hogy töröljük a MODOSITO és a KOLTSEGVISELOK állományt, majd az UJALLOMANY-t átnevezzük KOLTSEGVISELOK-re. Ezzel előállítottuk a minden menet számára elfogadható környezetet. Akár nyomtatni, akár lekérdezni stb. akarunk, megtehetjük. Sőt újabb karbantartó menetet is indíthatunk, ha előbb létrehozunk egy új MODOSITO állományt.
Bonyolítja a helyzetet, ha van TARTALEK állományunk is, és azt akarjuk, hogy az mindig az aktuális KOLTSEGVISELOK állomány tartalmát tükrözze (35. ábra).

Ilyenkor törölnünk kell a MODOSITO, a KOLTSEGVISELOK és a TARTALEK állományt, majd az UJALLOMANY-t át kell neveznünk KOLTSEGVISELOK-re, végül erről kell egy TARTALEK állományt másolnunk.
Előfordulhat, hogy e műveletek közben, valamilyen hiba következtében az állományok megsemmisülnek. A műveletek megfelelő sorrendben való elvégzésével azonban elkerülhető, hogy az ilyen hibák katasztrófát okozzanak, azaz hogy arra kényszerüljünk, hogy a teljes törzsállományt újra előállítsuk, vagy hogy a teljes feldolgozást újra végrehajtsuk. (Ez nem is mindig lehetséges.)
Ha például először csak a KOLTSEGVISELOK állományt töröljük, majd az UJALLOMANY-t átnevezzük KOLTSEGVISELOK-re, az állományok megsemmisülése esetén a KOLTSEGVISELOK a TARTALEK-ból előállítható, és mivel a MODOSITO még megvan, csak a karbantartást kell megismételnünk, és előáll az UJALLOMANY is.
Ha viszont sikerült a művelet, a KOLTSEGVISELOK-ről másolhatunk egy UJTARTALEK-ot. Ha ez nem sikerül, sőt a KOLTSEGVISELOK is megsérül, a meglevő állományokból a fenti módon természetesen újra előállítható.
Ha elkészült a hibátlan UJTARTALEK, törölhetjük a MODOSITO és a TARTALEK állományt, majd átnevezhetjük az UJTARTALEK-ot TARTALEK-ra. Hiba esetén a meglevő állományok itt is lehetővé teszik a hibátlan állapot helyreállítását.
Jól utánagondolva látható, hogy itt valójában az állományok megduplázásának az elvét követjük, ha csak ideiglenesen is. Addig nem törlünk egy régi állományt, amíg egy sikeres művelet eredményeként az új állományról nem állítottunk elő egy hibátlan, használható másolatot.



35. ábra. „Rendcsinálás" az állományoknál

Erre a túlzott óvatosságra elsősorban akkor van szükségünk, ha a hardver nem tökéletesen megbízható, vagy ha az állományunk – például az újraelőállíthatatlansága miatt – különösen értékes.

A kérdés mármost az, hogy kire bízzuk a rendcsinálás kényes feladatát? Az eddigi példáinkban ez a gépkezelő kötelessége volt. Ezt a funkciót a SEQREND rendszer sem vette át tőle, csak segítséget nyújtott a kényelmesebb végrehajtásához.

Az operátori rendcsinálás előnye, hogy szükségállapotban, azaz hiba esetén a gépkezelő azonnal reagálhat, és megfelelő beavatkozással minimalizálhatja a hiba által okozott kárt. Ehhez persze jól képzett és értelmes operátorra van szükség. Hátránya viszont, hogy maga az operátor is követhet el hibát, például tévedésből, figyelmetlenségből, elgépelésből kifolyólag törölhet olyan állományt, amire még szükség van.

Az operátor tévedése kizárható, ha a rendcsinálást maguk a programok hajtják végre, vagyis a megfelelő műveletek a programokba be vannak építve. Az így automatizált rendcsinálás igen kényelmes, de hátránya, hogy azok a hibák, amelyek nem járnak sem prog-

rammegszakítással, sem a képernyőre kiadott hibaüzenettel, rejtve maradhatnak, sőt átgyűrűzhetnek a további feldolgozásba, és esetleg csak több menettel később ismerhetők fel, amikor a hatásuk már többnyire végzetes. Ez ellen csak a programot bonyolító és növelő intenzív hibavizsgálattal védekezhetünk. (Így e módszer leginkább ott alkalmazható, ahol a hardver nagy üzembiztonsággal működik.)

Ha nem kell rendszeresen hibára számítanunk, de annak lehetősége mégsem kizárt, egy hibrid megoldást javasolunk. Írjunk egy rendcsináló programot vagy rutint, amely megfelelően paraméterezhető. Ebbe legyen beépítve a hibavizsgálat. A feldolgozó programok ezt a rutint hívják, a rendcsináláshoz szükséges paraméterek megadásával. A rutin pedig végrehajtja a szükséges műveleteket, és figyeli azok sikerességét. Ha minden rendben van, nem jelez, dolgozik tovább, végül visszaadja a vezérlést a hívó modulnak. Ha viszont hibát észlel, operátori beavatkozást kér.

Természetesen nem kell általános rendcsináló rutint írnunk, elegendő olyan, amely az adott rendszer igényeit kielégíti. Például a költségviselőket nyilvántartó rendszerben ilyen rutinra — ha nem kívánunk TARTALEK állománnyal dolgozni — kizárólag a SEQKARB vagy SEQMODO programban van szükség. Tartalék állomány kezelése esetén is csak még egy program, a SEQFELV jön számításba.

```
2000 REM
2001 REM     ----- RENDCSINALAS -----
2002 REM
2003 REM     MODOSITAS [SEQMODO] UTAN
2004 REM
2005 REM     HIVASA:
2006 REM
2007 REM     999 GOSUB 2010 : END
2008 REM
2009 REM
2010 OPEN 15,8,15
2020 UZ$="KOLTSEGVISELOK TORLESE:"
2030 PRINT#15,"SCRATCH:KOLTSEGVISELOK"
2040 INPUT#15,H,H$
2050 IF H>19 THEN GOTO 2110
2060 UZ$="MODOSITOTT ATNEVEZESE:"
2070 PRINT#15,"RENAME:KOLTSEGVISELOK=MODOSITOTT"
2080 INPUT#15,H,H$
2090 IF H>19 THEN GOTO 2110
2099 CLOSE 15 : RETURN
2110 PRINT
2120 PRINT UZ$;H;H$
2199 CLOSE 15 : STOP
```

GYORSÍTOTT KARBANTARTÁS

A SEQKARB program nagyon lassú, hiszen annyiszor kell végigolvasnunk a MODOSITO állományt, csupán a keresés céljából, ahány rekord van a törzsben.

Nagyságrenddel meggyorsítható a karbantartás, ha a keresést nem a lemezen, hanem a tárban hajtjuk végre. Vagyis a MODOSITO állomány rekordkulcsait beolvassuk egy alkalmas tömbbe, és a továbbiakban ott keresünk, mintha az volna a módosító állomány.

Induljunk ki a SEQKARB karbantartó programból, és módosítsuk azt ennek megfelelően. Mindenekelőtt vegyünk fel egy vektort a módosító rekordok azonosítóinak:

```
30 DIM KK$(20)
```

Amikor a MODOSITO állományt először végigolvassuk, és átvisszük belőle az UJALLOMANY-ba a nem törlendő rekordokat, akkor a módosító rekordok K$ azonosítóját mentsük ki a KK$ vektorba:

```
130 I=0
240 IF M$="TORLENDO" THEN GOTO 260
260 I=I+1:KK$(I)=K$
315 KK$(I+1)=VEGE$
```

Minthogy a továbbiakban nem a MODOSITO állományban, hanem a KK$ vektorban keresünk, az állományt többé nem kell használnunk. A megnyitása helyett viszont a vektor indexét kell visszaállítani a kezdőértékre, azaz nullára, hogy a keresés mindig elölről kezdődjön:

```
440 I=0
```

A MODOSITO állomány olvasása pedig teljesen elmarad. Helyette csak az indexet kell léptetni, azaz megnövelni 1-gyel:

```
510 I=I+1
520 <-- TORLENDO
```

Persze a rekordazonosítót minden előfordulásakor ezzel kell indexelni:

```
530 IF KK$(I)=VEGE$ THEN GOTO 720
540 IF KK$(I)=K$    THEN GOTO 410
```

Az állomány lezárását is értelemszerűen el kell hagynunk:

```
610 <-- TORLENDO
699 <-- TORLENDO
710 <-- TORLENDO
```

A módosított karbantartó program végülis a 174–175. oldalon látható.

Ha van kedvünk hozzá, futtassuk le a karbantartás mindkét változatát, és mérjük le az időkülönbséget. (Mi 7 törzs- és 7 módosító rekorddal 35 helyett 10 másodpercet mértünk, ami valamivel több, mint 70%-os időmegtakarítást jelent.)

A program arra jó példa, hogy miként kell egy *programot optimalizálni.* Az *első szabály* az, hogy hibás programot nem optimalizálunk. A *második szabály* pedig, hogy csak akkor optimalizálunk, ha az feltétlenül szükséges. Végül a *harmadik szabály:* találjuk meg a programnak azt az 5%-át, amely a futási idő 95%-áért felelős, és ezen változtassunk, mégpedig soha ne becslés, hanem mindig csak mérések alapján.

```
1 PRINT "♥"
2 PRINT
3 PRINT " KOLTSEGVISELOK MODOSITASA"
4 PRINT " -------------------------"
5 PRINT
6 PRINT
7 INPUT " MEHET =I/N=";V$
8 IF V$<>"I" THEN GOTO 999
10 VEGE$="ππππ"
20 S$=CHR$(13)
30 DIM KK$(20)
100 REM: ------------------------------ MEGNYITASOK
101 REM:                                -----------
110 OPEN 2,8,2,"UJALLOMANY,SEQ,WRITE"
120 OPEN 3,8,3,"MODOSITO,SEQ,READ"
130 I=0
200 REM: ---------------------------MODOSITO ATVITELE
201 REM:                            -----------------
210 K$=VEGE$
220 :  INPUT#3,K$,M$,T,R
230 :  IF K$=VEGE$        THEN GOTO 310
240 :  IF M$="TORLENDO" THEN GOTO 260
250 :  PRINT#2,K$;S$;M$;S$;T;S$;R
260 :  I=I+1:KK$(I)=K$
299 GOTO 210
300 REM: -----------------------------TORZS ATVITELE
301 REM:                              --------------
310 CLOSE 3
315 KK$(I+1)=VEGE$
320 OPEN 4,8,4,"KOLTSEGVISELOK,SEQ,READ"
400 REM: ------------------------------------ATVITEL
401 REM:                                     -------
410 K$=VEGE$
420 :  INPUT#4,K$,M$,T,R
430 :  IF K$=VEGE$ THEN GOTO 810
440 :  I=0
500 REM: ------------------------------------KERESES
501 REM:                                     -------
510 :  I=I+1
530 :  :  IF KK$(I)=VEGE$ THEN GOTO 720
540 :  :  IF KK$(I)=K$    THEN GOTO 410
599 :  GOTO 510
700 REM: ------------------------------ HA NEM TALALTA
701 REM:                                -------------
720 :  PRINT#2,K$;S$;M$;S$;T;S$;R
```

```
799 GOTO 410
800 REM:------------------------------------------ZARASOK
801 REM:                                           -------
810 CLOSE 4
820 CLOSE 2
900 REM:---------------------------------------BEFEJEZES
901 REM:                                       ---------
910 PRINT
920 PRINT " MODOSITAS KESZ"
930 PRINT
940 PRINT " ADATOK AZ ■UJALLOMANY■-BAN"
950 PRINT
999 END
```

Megjegyezzük, hogy a programunk legfeljebb 20 rekordból álló módosító állományt tud kezelni, mert ekkorára definiáltuk a vektort. Noha ez még messze van a gépi korláttól, és természetesen növelhető, a tár kapacitásának végessége miatt nem tarthatók így karban olyan állományok, amelyeknél a módosítások várható száma sok. A teszteléshez azonban ennyi is elég.

További érdekesség, hogy nincs szükségünk a teljes módosító rekord tárolására. Ezért nem vettünk fel egy-egy vektort az MM$, TT, RR adatok számára is. A kereséskor ugyanis már csak a rekord puszta léte vagy nem léte a fontos számunkra, az adatai érdektelenek – azokat ugyanis már felhasználtuk.

Utószó

A soros állományok kezelésének témáját csak abbahagyni lehet, befejezni nem. Mindig jöhetnek elő újabb és újabb megoldások, ötletek, fogások, változatok. Nincs két egyforma környezet, nincs két egyforma feladat, nincs két egyforma megoldás. Éppen ebben rejlik az adatfeldolgozás szépsége.
Ennek megfelelően a bemutatott programjaink nem arra valók, hogy lemásoljuk őket, hanem arra, hogy gondolatokat ébresszenek bennünk, ha soros adatállományokat kell kezelnünk.
Azzal búcsúzunk, hogy továbbra is szívesen várjuk olvasóink véleményét és észrevételeit, a már ismert címen:

INTRONIK Számítástechnikai és Elektronikai Műszaki Fejlesztő Kisszövetkezet
Budapest, Pf. 348.
1 4 4 5

Irodalom

Angerhausen–Becker–Gerits–Schellenberger: A BASIC programozás magasiskolája a C64-esen. DATA BECKER – NOVOTRADE, Budapest, 1985.
Bakó András dr.: Commodore 64 mikrogép programozása. Budapest, 1983.
Bodor Tibor: Commodore 64 felhasználói segédlet. INTRONIK, Budapest, 1984.
Bodor Tibor: Commodore 64 lemezkezelés (hallgatói segédlet). INTRONIK, Budapest, 1984.
Bodor Tibor–Gerő Péter: A BASIC programozás technikája. SZÁMALK, Budapest, 1983.
Bodor Tibor–Gerő Péter: A Commodore 64 programozásának gyakorlata. ALAPISMERETEK. SZÁMALK, Budapest, 1985.
Bodor Tibor–Gerő Péter: Bevezetés a korszerű FORTRAN programozásba. SZÁMALK, Budapest, 1983.
Bodor Tibor–Gerő Péter: MIKROSZÁMÍTÓGÉPEK felhasználói segédkönyv az azonos című oktatófilm-, videokazetta- és diaképsorozathoz. NOVOTRADE – DELTASOFT, Budapest, 1985.
COMMODORE GmbH: Commodore 64 Bedienungshandbuch. Frankfurt
COMMODORE GmbH: Commodore 64 MicroComputer Handbuch. Frankfurt
COMMODORE GmbH: Floppy Disk VC 1541 Bedienungshandbuch. Frankfurt
COMMODORE GmbH: VC 1541 Floppy Disk Bedienungs Handbuch. Frankfurt
COMMODORE Inc.: VIC–1541 Floppy Disk User's Manual. 1982.
COMMODORE Ltd.: Commodore 1541 Disk Drive User's Guide. Westchester, 1982.
Gerő Péter: Commodore 64 BASIC (hallgatói segédlet). INTRONIK, Budapest, 1984.
Lángos István: A Commodore 64 mikrogép kezelése és programozása. COMPORGAN, Budapest
NOVOTRADE Rt.: Commodore 64 felhasználói kézikönyv. Budapest
Ury László dr.: Commodore 64 BASIC felhasználói kézikönyv. LSI ATSZ, Budapest, 1984.
Ury László dr.: Commodore 64 információs kártya. LSI ATSZ, Budapest
Commodore VIC–1541 floppy disk felhasználói kézikönyv.

Tárgymutató

Tárgymutatónk csak azokat a fogalmakat tartalmazza, amelyek nem szerepelnek az „ALAPISMERETEK" kötetben. Ezenkívül megadjuk az angol szavak szokásos kiejtését és jelentését, valamint utalunk arra, hogy hol találhatunk róluk részletesebb információt.

A

adatállomány **73**
— átnevezése **13, 18, 71, 99, 100**
— felülírása **16, 17, 110**
— másolása **13, 18, 67, 71, 93, 94**
— törlése **13, 18, 71, 93, 94**
adat bekérése **82, 83**
adatcsatorna **22, 23**
adat ellenőrzése **83**
adat javítása **85**
adatkezelés **28**
— numerikus adatok **32, 33, 39**
— karakteres adatok **33, 34, 39**
állomány **22, 23, 41**
— írása **25, 26, 34, 35, 37, 39, 40**
— lezárása **25, 26, 27, 48**
— megnyitása felülírásra **26, 27**
— megnyitása írásra **24, 26**
— megnyitása olvasásra **24, 26, 27, 28, 30**
— megnyitása továbbírásra **25, 26**
— olvasása adatonként **25, 26, 27, 30, 40**
— olvasása bájtonként **25, 28, 40**
— olvasása rekordonként **35, 38, 40**
— végének figyelése **29, 31, 57, 87, 89, 91, 113, 143**
— ⇒ soros állomány
állományazonosító **22**

állománykezelő utasítások **24**
állománynév **16, 22, 24, 28, 47**
állományvége jel **29, 57**
APPEND (apend): hozzáfűz, hozzátold = állomány megnyitása továbbírásra
archív állományok **93, 170**
átállási tevékenység **140**
átnevezés ⇒adatállomány/program

B

beszúrás ⇒karbantartás
betöltés ⇒program
BLOCKS FREE (bloksz fri): szabad blokkok = tartalomjegyzék
blokk **14, 21, 29, 45, 70**
bővítés ⇒soros állomány

C

ciklikus hívás **132**
ciklus
– elöltesztelő **112, 118, 166**
– hátultesztelő **91**
CLOSE (klóz): becsuk, lezár ⇒lezárás
COPY (kopi): másol, másolat ⇒másolás
csere ⇒karbantartás
csoportfeltétel ⇒csoportképző
csoportképző
– adat **139**
– feltétel **140, 141**
változó **139**
csoportkezdő tevékenység **140**
csoportonkénti feldolgozás **138**
csoportváltás **140**
csoportváltozó ⇒csoportképző
csoportzáró tevékenység **140**

D

DEVICE NOT PRESENT (divájsz not prezent): az eszköz nem található ⇒lemezegység
DIRECTORY ERROR (direktori eror): hibás tartalomjegyzék ⇒hibaüzenetek
DISK FULL (diszk ful): lemez megtelt = hibaüzenetek
DRIVE NOT READY (drájv not redi): a lemezegység nem üzemkész ⇒hibaüzenetek

E

egymásba ágyazott csoportok **140, 141, 144**
elérési mód **73**
elhatároló jelek **22, 25, 28, 33, 34, 36, 37, 39**
ellenőrzés ⇒ program
elöltesztelő ciklus ⇒ciklus
EOF=end of file (eoef, end of fájl): állomány vége

F

feldolgozási környezet **169, 171, 172**
feldolgozási menetek **168**
feldolgozó tevékenység **140**
FILE DATA ERROR (fájl déjta eror): adathiba az állományban ⇒adatkezelés
FILE EXISTS (fájl egziszc): az állomány már létezik ⇒ hibaüzenetek
FILE NOT FOUND (fájl not fáund): az állomány nem található ⇒hibaüzenetek
FILE NOT OPEN (fájl not ópen): az állomány nincs megnyitva ⇒hibaüzenetek
FILE OPEN (fájl ópen): az állomány már nyitva van ⇒ megnyitás
FILES SCRATCHED (fájlz szkercst): állományok törölve ⇒hibaüzenetek
FILE TYPE MISMATCH (fájl tájp mizmecs): az állomány típusa nem megfelelő ⇒hibaüzenetek
fix rekord ⇒rekordok kezelése
formázás ⇒lemez
funkciók kiválasztása **156**

G

GET (get): beszerez, bevesz ⇒ adatkezelés/állomány olvasása
gyorsított karbantartás **173**

H

hátultesztelő ciklus ⇒ciklus
helyi érték szerinti kiírás **143**
hibaállapot lekérdezése **27, 28, 70, 97, 98**
hibaüzenetek **71**
HIGH VALUE (háj veljú): legmagasabb érték

I

INITIALIZE (inisiölájz): kezd, kezdőértéket beállít ⇒ lemezegység állapotának helyreállítása
INPUT (input): bemenet, bevitel ⇒ adatkezelés/állomány olvasása

interaktív módosítás **104**
írás
— adatállományba ⇒állomány írása
— nyomtatóra ⇒nyomtató használata
— parancs-csatornára ⇒parancs-csatorna használata
ismétlődő tevékenységek **132**, ⇒ciklus

K

karakteres adatok ⇒adatkezelés
karbantartás **126, 131, 138, 160, 173**
képernyő kezelése **49, 82, 83**
keresés
— rendezetlen soros állományban **112, 161**
— rendezett soros állományban **124**
keretrendszer **154, 159**
kétlemezes funkciók **67, 71**
kilistázás ⇒ program/tartalomjegyzék
kimentés ⇒ program
kontrollfeltétel ⇒csoportképző
kontrollszakítás ⇒csoportonkénti
kontrollváltás ⇒csoportonkénti
kontrollváltozó ⇒csoportképző
közvetlen elérésű állományok **74**

L

lekérdezés ⇒soros állomány
lemez
— formázása **13, 14, 67, 71**
— nevének ellenőrzése **50**
— szerkezete **43**
— tartalomjegyzéke ⇒tartalomjegyzék
— tömörítése **13, 21, 71**
— újraformázása **13, 14**
lemezazonosító **13, 14, 45**
lemezegység
— állapotának helyreállítása **13, 21, 22**
— egységszáma **22, 68**
— számának átállítása **69**
lemezfej **46, 50**
lemezkezelő parancsok **13, 22**
lemezműveletek **13, 22, 41**
lemeznév **13, 14, 45, 50**
lemeztérkép **19, 20, 44, 45, 64**
létrehozás ⇒soros állomány
lezárás ⇒állomány/nyomtató/parancs-csatorna

LIST (liszt): lista, listáz ⇒kilistáz
LOAD (lód): betölt ⇒program betöltése
LOADING (lóding): betöltés = program betöltése

M

másolás ⇒adatállomány/program
megnyitás ⇒állomány/nyomtató/parancs-csatorna
módosítás ⇒soros állomány

N

NEW (nyú): új ⇒ lemez formázása/program törlése tárból
NO CHANNEL (no csenöl): nincs szabad adatcsatorna ⇒ hibaüzenetek
numerikus adatok ⇒adatkezelés
nyomtatás ⇒soros állomány
nyomtató
— használata **17, 89**
— lezárása **17, 89**
— megnyitása **17, 88**

O

OK (oké): minden rendben ⇒program ellenőrzése/hibaüzenetek
OPEN (ópen): nyit, nyitás ⇒megnyitás
összefésülés = rendezett állományok

P

parancs-csatorna **13, 22, 23, 24**
— használata **13, 14, 18, 22, 69**
— lezárása **14, 18, 23, 27, 41, 69**
— megnyitása **13, 14, 18, 22, 27, 41, 69**
— olvasása **27, 31, 32, 70, 97, 172**
párosítás **124, 125, 129**
PRINT (print): nyomtat, nyomtatás ⇒írás
program
— átnevezése **13, 18, 71**
— betöltése **13, 17, 71**
— betöltése programból **156, 157, 158**
— ellenőrzése **13, 15, 71**
— felülírása **16, 17**
— kilistázása **15**
— kimentése **13, 14, 15, 16, 71**
— kinyomtatása **17**
— másolása **13, 17, 18, 67, 71**
— optimalizálása **173**

– törlése lemezről **13, 18, 67, 71**
– törlése tárból **14**
programnév **15, 16, 47, 56**
programrendszer **153**
puffer **87, 91**

R

random adatállomány **73**
READ (rid): olvas, beolvas ⇒ állomány megnyitása olvasásra
READ ERROR (rid eror): olvasási hiba ⇒ hibaüzenetek
RECORD NOT PRESENT (rekord not prezent): a rekord nem található ⇒ hibaüzenetek
rekord **34**
– felvitele **84**
– olvasása **89**
– törlése **101, 102, 130**
rekordok kezelése **34, 84, 89**
– fix hosszúságú **34, 35**
– változó hosszúságú **36, 37**
rekordok párosítása **124, 125, 129**
rekordonkénti feldolgozás **38**
rekordonkénti módosítás **125, 160**
relatív adatállomány **74**
RENAME (rinéjm): új nevet ad valaminek ⇒ átnevezés
rendezetlen karbantartás **160, 173**
rendezett állományok **123**
– karbantartása **124, 126, 131**
– összefésülése **147**
rendezett felvitel **133**
rendezési elv **123**
RETURN (ritörn): visszatér, visszküld ⇒ sorvége jel
RUN (rán): fut, futás

S

saját végjel **81, 81**
sáv **43**
– sávcím **43, 62, 70**
SAVE (széjv): megment, tartalékol ⇒ program kimentése
SAVING (széjving): kimentés ⇒ program kimentése
SCRATCH (szkrecs): kivakar, levakar, töröl ⇒ törlés
SEARCHING (szörcsing): keresés, kutatás ⇒ program betöltése
SEQ=sequential (szeku, szikvensöl): soros, szekvenciális ⇒ állomány megnyitása
soros adatállomány **73, 77**
soros állomány
– átnevezése **99, 100**

— bővítése **92, 95**
— feldolgozása **87**
— felülírása **110**
— írása **84**
— karbantartása tömbökkel **173**
— kinyomtatása **87, 88**
— lekérdezése **110**
— létrehozása **80**
— lezárása **84, 89**
— másolása **93**
— megnyitása írásra **81**
— megnyitása olvasásra **89**
— megnyitása továbbírásra **92**
— mérete **78**
— módosítása **104, 160, 173**
— olvasása az állomány vége után **91**
— olvasása rekordonként **89**
— összemásolása **93, 94**
— rekordjai **78, 80**
— rendezett létrehozása **133**
— szűkítése **101**
— törlése **96, 97**
— válogatása **118**
— ⇒állomány
soros feldolgozás **77, 166**
soros keresés **112, 161, 166**
sorvége jel ⇒elhatároló jelek
STATUS (sztéjtösz, státusz): állapot ⇒ állomány végének figyelése
SYNTAX ERROR (szinteksz eror): helyesírási hiba ⇒ hibaüzenetek
szektor **43**
— szektorcím **44, 62, 70**
szeparátorjel ⇒elhatároló jelek
szervezési mód **73**
szűkítés ⇒soros állomány

T

tartalomjegyzék **14, 19, 20, 21, 45, 46, 61, 64**
— bejegyzései **45**
— bejegyzéseinek elérése **61**
— bejegyzéseinek tartalma **47**
— betöltése **14**
— kilistázása **14**
— kinyomtatása **17**

— olvasása **50, 51, 56**
tárolási mód **73**
tételenkénti módosítás **104**
típuskód **47, 48**
több állomány kezelése **101, 128**
többfokozatú csoportváltás **144**
több lemezegység **68**
tömbök **173**
tömörítés ⇒ lemez
törlés ⇒ adatállomány/program

U

újraformázás ⇒ formázás
üres állomány **25, 31**
üzenet villogtatása **84**

V

VALIDATE (velidéjt): érvényesít, megalapoz ⇒ lemez törlése
válogatás = soros állomány
változó rekord = rekordok kezelése
végső csoport **141**
VERIFY (verifáj): ellenőriz, bizonyít ⇒ program ellenőrzése
VERIFY ERROR (verifáj eror): az ellenőrzés sikertelen ⇒ program ellenőrzése
VERIFYING (verifájing): ellenőrzés, egybevetés ⇒ program ellenőrzése
vezérlő jelek **49**
vezérlő program **154**

W

WRITE (rájt): ír, felír ⇒ állomány megnyitása írásra
WRITE ERROR (rájt eror): íráshiba ⇒ hibaüzenetek
WRITE FILE OPEN (rájt fájl ópen): az állomány írásra van megnyitva ⇒ hibaüzenetek
WRITE PROTECT ON (rájt protekt on): írásvédelem bekapcsolva ⇒ hibaüzenetek

Tibor S. Bodor

SEQUENTIAL DISK FILES

The book has been written for those who are just getting to be acquainted with disk files and data processing on the COMMODORE 64.

In Part I, the disk handling commands are overviewed emphasizing their effects on the sequential files. The physical organization of the disk, the directory and the block availability map are thoroughly described here. Three programs are included for demonstrating how to access characteristic information stored in the entries of the directory.

In Part II, the theory and the practice of sequential file processing are presented. All of the main file handling functions, such as generating, processing, updating, deleting, renaming, rewriting, searching, copying, concatenating, appending, mergeing, even control breaks are discussed.

Programming techniques are illustrated by full programs that can be executed on the didactically designed test files provided in the book.

Kiadja: a Számítástechnika-alkalmazási Vállalat
Felelős kiadó: Havass Miklós vezérigazgató
Felelős szerkesztő: Dr. Brückner Huba
Műszaki vezető: Molnár Zoltán
Műszaki szerkesztő: G. Müller Zsuzsa
A fedelet tervezte: Molnár Zoltán
Az ábrákat rajzolta: Zeikfalvi Lenke
Megjelent: 17,25 (A/5) ív terjedelemben

87/964 – Vízügyi Dokumentációs Szolgáltató Leányvállalat nyomdája
Bp., VII., Kazinczy u. 3/b
Felelős vezető: Kaj István

Ára: 133,-Ft